UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO SMANIA SPILLERE

CRIAÇÃO DE COMPONENTES DINÂMICOS PARA JOOMLA POR MEIO DE UML

CRICIÚMA, JULHO DE 2008

UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO SMANIA SPILLERE

CRIAÇÃO DE COMPONENTES DINÂMICOS PARA JOOMLA POR MEIO DE UML

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense – UNESC.

Orientador: Prof. MSc. Eduardo Menna da Silva

CRICIÚMA, JULHO DE 2008

RODRIGO SMANIA SPILLERE

CRIAÇÃO DE COMPONENTES DINÂMICOS PARA JOOMLA POR MEIO DE UML

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Profa. MSc. Ana Claudia Garcia Barbosa

Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

Prof. MSc. Eduardo Menna da Silva (UNESC)

Orientador

Prof. MSc. Ana Claudia Garcia Barbosa (UNESC)

Prof. MSc. Cristiane Raquel Woszezenki (UNESC)

À meu pai Dilnei Spillere, pelo exemplo de perseverança e amor ao próximo em sua luta por uma nova vida.

AGRADECIMENTOS

Em primeiro lugar a Deus e as boas forças do universo, por permitirem a minha evolução pessoal e espiritual.

Aos meus pais Dilnei e Mari, pela minha educação, princípios, formação como cidadão e principalmente por acreditarem em meu potencial desde o princípio.

À minha amada namorada Thais, por me “aturar” em mais essa trajetória no mundo acadêmico e me dar a maior força em todos os momentos de estresse que envolveu o desenvolvimento deste trabalho.

À meu orientador Eduardo Menna da Silva, por sua dedicação, disponibilidade e responsabilidade a cerca de nosso trabalho.

Aos amigos que apoiaram e me ajudaram a esclarecer idéias e dúvidas ao longo do trabalho. Em especial a Dirceu Pereira Tiegs por me mostrar alguns afluentes da minha pesquisa e Diego Burigo Zacarão por sanar algumas dúvidas com relação a classe SimpleXMLElement.

Ao pessoal do Joomla Team, por apoiarem minha pesquisa e ceder um espaço para eu submeter alguns artigos a comunidade.

À todas as pessoas que de alguma maneira contribuíram para este sonho se tornar realidade.

“... Os grandes navegadores devem sua fama

aos temporais e tempestades ...”

(Epicuro de Samos)

RESUMO

CRIAÇÃO DE COMPONENTES DINÂMICOS PARA JOOMLA POR MEIO DE UML

Este trabalho tem por objetivo extrair as informações dos meta-modelos de diagramas de classes UML e, por meio desses, utilizar técnicas de *Model Driven Architecture* (MDA) para gerar a estrutura de códigos fonte dos componentes para o ambiente Joomla.

Criar componentes para Joomla requer do programador um estudo aprofundado de sua API. Não basta saber programar na linguagem PHP na qual ele é escrito. É necessário também, estudar suas classes, objetos, métodos de acesso aos dados entre outros.

Para abstrair tais padrões, foi desenvolvida uma ferramenta em PHP que visa manter as boas práticas de engenharia de software e, a partir desta, possibilitar a geração automática da arquitetura dos componentes para o f*ramework* Joomla. A ferramenta traduz o arquivo XMI proveniente dos diagramas de classes UML e efetua a geração de códigos por meio de *parsings* XML.

Desta maneira, abstrai-se do programador grande parte do código que se destina a padronização, criação de estruturas e bases de dados. Acarretando na redução de tempo de programação, abstração da complexidade do código do componente e reutilização de códigos.

Palavras-chave: Engenharia de Software, Desenvolvimento Web, Gerenciadores de Conteúdos, Joomla, Componentes Joomla, MDA.

ABSTRACT

CREATION OF DYNAMICS COMPONENTS FOR JOOMLA BY MEANS OF UML

This work has for objective to extract informations from meta-models of category UML diagrams and through that, make use of Model Driven Architecture (MDA) technics to generate source codes structure of components for Joomla.

Create Joomla components request to programmer make a profound study of API. It's not sufficient know to program PHP language in wich Joomla was written. Besides it's need, study their categories, objects, ways to dates access among other things.

To abstract these models, was developer the PHP tool to aim at support the good practice of software engineering and, as of this, to enable automatic generation of components architecture for framework Joomla. The tool translate the XMI file deriving from category UML diagram and effectuate the code generation by parsings XML.

So that, abstract of programmer high code piece data bases. Resulting of reduction programming time, abstraction of complexity component code and make useful again the codes.

Key-words: Software engineering, Web development, contents managements, Joomla, Joomla components, MDA.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API - Aplication Program Interface

CMS - Content Management System

GPL - General Public License

MDA - Model Driven Architecture

MOF - Managed Object Format

MOS - Mambo Open Source

MVC - Model View Controller

OMG - Object Management Group

PC - Personal Computer

PDA - Personal Digital Assistant

PHP - PHP Hypertext Processor

SGC - Sistemas de Gestão de Conteúdos

SGDB - Sistema Gerenciador de Bancos de Dados

SQL - Structured Query Language

UML - Unified Modeling Language

XMI – XML Metadata Interchange

XML - Extensible Markup Language

W3C - World Wide Web Consortium

WCM - Web Content Management System

WSGC - Sistemas de Gestão de Conteúdos Web

SUMÁRIO

1 INTRODUÇÃO

Observando a linha do tempo e a história, é possível perceber que alguns bens serviram de base para a economia. A propriedade, a mão de obra, as máquinas e o capital são exemplos desses bens. Hoje, a economia gira em torno da informação e, por conta disso, quanto mais informação a empresa possui, maiores serão as vantagens em relação às outras. Em conseqüência do crescimento das informações, surgiu-se à necessidade de gerenciá-las de uma maneira mais simplificada. Por este motivo, foram criados *Content Management System* (CMS), onde o emprego mais popular se refere a: Gerenciador de Conteúdos Web, ou seja, é uma combinação de ferramentas que auxiliam o usuário na manutenção de seu site, portal ou sistema via Web (GRAF, 2006). Existem no mercado inúmeros CMS disponíveis na Web. Em software livre, existem os escritos na linguagem PHP, como por exemplo: PHP-Nuke, XOOPS, Mambo, Joomla, entre outros.

O Joomla é uma ferramenta que auxilia leigos a criar seus próprios portais (JOOMLA BRASIL, 2007). Porém, esta acaba se tornando complexa na resolução de problemas que utilizam customizações em seu código fonte. Como exemplo, é possível citar o uso de componentes que contenham lógicas de negócios, a criação de novos módulos personalizados e o desenvolvimento de *plugins* e utilitários. Ao programador, isso implica na necessidade de conhecimentos avançados tanto na linguagem de programação em que o CMS foi escrito, quanto de sua *Application Programming Interface* (API). Esse cenário normalmente torna o trabalho bastante cansativo, pois prolonga o tempo de desenvolvimento de seu projeto em função de colocá-lo nos padrões requeridos. Por outro lado, força o desenvolvedor a criar ferramentas padronizadas e melhor estruturadas. Tal inconveniente pode acarretar no abandono da

ferramenta para projetos que necessitam de lógica de negócios em seus componentes, pelo fato da complexidade que o mesmo não trata, ocasionando, assim, atrasos na implantação do mesmo.

O Joomla é um CMS construído por meio de tecnologias em Software Livre, capaz de criar e gerenciar portais completos. Assim, não há necessidade do usuário ter conhecimentos de programação para utilizar os seus recursos pré-programados (GRAF, 2007).

Em um sistema com muitas classes e heranças interagindo todo o tempo com milhares de componentes encapsulados em objetos, caso seja preciso buscar a procedência dos mesmos, é necessário, no mínimo, abrir muitos arquivos e pesquisar de onde vieram as classes que originaram os objetos os quais estão sendo procurados. Com o advento da *Unified Modeling Language* (UML), os mesmos podem ser encontrados rapidamente seguindo um diagrama visual, onde é possível ter uma visão panorâmica do projeto, além de auxiliar em diversos setores como: criação de novos sistemas, compreensão de complexidades de sistemas já programados, facilidade na documentação, entre outros.

Pensando nisso, este trabalho propõe a criação de uma ferramenta que auxilie o programador de maneira que, a partir da estruturação de seu programa por meio da UML, possibilite a geração de códigos que sigam o padrão estipulado pela equipe de desenvolvimento do Joomla. Desta forma, é possível abstrair grande parte da complexidade da programação de componentes para Joomla e incentivar os programadores a construir seus componentes estruturados em uma linguagem de modelagem universal.

1.1 OBJETIVO GERAL

Permitir a geração automática da estrutura de códigos que compreende o padrão dos componentes do framework *Joomla* por meio de diagramas de classes UML, a fim de auxiliar o programador na tarefa da confecção de componentes personalizáveis.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são os seguintes:

a) compreender o funcionamento de componentes para Joomla, modelagem de Diagramas de Classes UML e captura das informações geradas por estes diagramas;

b) desenvolver a ferramenta para a captura e conversão dos dados dos diagramas;

c) converter o código gerado pela ferramenta para o padrão da arquitetura do CMS Joomla;

d) incluir na ferramenta métodos genéricos de cadastro, busca e visualização, a fim de auxiliar o programador na tarefa de criação de componentes Joomla;

e) instalar o componente gerado pela ferramenta no servidor que possui o CMS Joomla;

f) disponibilizar a ferramenta sob Licença de Software Livre *General Public Licence* (GPL) para que os usuários possam utilizar e contribuir com o desenvolvimento da mesma.

1.3 JUSTIFICATIVA

Dentre as inúmeras vantagens para se usar um CMS, descritas por Graf (2006), como o tempo de desenvolvimento, estabilidade, segurança, controle e custo, pode-se perceber o quão fácil é utilizar Sistemas Gerenciadores de Conteúdo, principalmente se o mesmo for mantido por uma comunidade com milhões de usuários como é o caso do CMS Joomla.

O CMS Joomla 1.5 é um *framework* em PHP orientado a objetos, seguindo o padrão universal *Model - View – Controler* (MVC). A partir desta API, pode-se criar qualquer tipo de aplicativo *web*, desde um *Blog* (Diário Virtual) até soluções comerciais complexas (GRAF, 2007).

O processo de criação de um software é basicamente separado em três etapas: (i) a etapa da modelagem do sistema efetuada pelo engenheiro de software que analisa o problema, constrói os diagramas e calcula os recursos necessários antes de começar a programar (DYKES; TITTEL, 2005); (ii) o desenvolvimento efetuado pelo programador; e, (iii) os testes que geralmente são executados pelos analistas ou pelos próprios usuários.

A reunião dessas etapas culmina em um processo que envolve muito conhecimento, investimento e recursos. Assim, a ferramenta proposta visa permitir que, ao invés de utilizar três passos e, conseqüentemente, três profissionais para personalizar um portal, seria preciso utilizar somente dois passos que poderiam ser executados pelo próprio programador da aplicação.

Assim esta pesquisa propõe o desenvolvimento de uma ferramenta que auxilie o programador, capacitando-o a personalizar um sistema via *Web* de maneira simples, abstraindo-o dos códigos usados para enquadrar os componentes em seu padrão.

Esta ferramenta consiste na tradução dos arquivos de saída dos diagramas de classes de ferramentas de modelagem gráfica UML que seguem o padrão XMI 1.1. Segundo Pender (2003), “este padrão permite a saída de informações UML das ferramentas de modelagens dos repositórios de objetos e a conversão para um formato padrão e transportável, por meio de XML”.

De posse disso, torna-se viável a construção de um aplicativo que leia os códigos XML gerados pela ferramenta de modelagem UML, desde que, sua linguagem escolhida possua suporte a leitura de arquivos XML e, logo após, realize a conversão para componentes Joomla.

Porém, para a utilização da ferramenta, é necessário que o programador cumpra alguns pré-requisitos. Um deles é o domínio da linguagem UML para a criação de seus componentes.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho foi dividido em 5 capítulos, conforme descrito a seguir.

O segundo capítulo apresenta os sistemas gerenciadores de conteúdos (CMS) e sua aplicação no dia-a-dia. O CMS Joomla, suas vantagens, casos de sucesso e utilização, também é parte integrante deste capítulo.

O terceiro capítulo aborda técnicas de engenharia de softtware especialmente UML, dando um enfoque especial aos diagramas de classes, que são fundamentais para a realização da proposta da presente pesquisa. Por fim, mostra um pouco sobre a técnica de converter os diagramas UML em código fonte, chamada *Model-Driven-Architecture* (MDA).

O quarto capítulo trata da ferramenta desenvolvida, o conceito, implementação, aplicações e resultados obtidos.

No ultimo capítulo é realizada a conclusão do presente trabalho e sugestões para pesquisas futuras.

2 SISTEMAS DE GESTÃO DE CONTEÚDOS

Com a célere evolução dos computadores pessoais e a popularização da Internet, a informação propagada por estes meios é consumida num ritmo cada vez mais veloz, de maneira que proporciona a qualquer pessoa, publicar conteúdos que podem ser visualizados em qualquer parte do planeta em questão de segundos. Segundo Mordeaui (2000), atualmente a informação é difundida em *bits* em vez de átomos. Não mais é preciso ir à banca de jornal para ficar atento aos acontecimentos do mundo, pois a mídia está disponível em diversas formas (PC, *LapTop*, PDA, entre outros).

À medida que a informação cresce, é imprescindível ter uma boa estratégia de gestão de conteúdos. Se, a cada notícia publicada nos grandes portais (como CNN[1], G1[2], Folha Online[3]), o *webmaster*[4] precisasse colocá-las estaticamente no portal, geraria um volume intenso de trabalho e seria necessária uma grande equipe técnica para executá-lo. Para resolver este problema, as empresas têm buscado soluções automatizadas para gerir seu conteúdo de forma que a informação possa ser produzida, revisada e publicada de uma maneira simples e instantânea, fazendo com que o próprio jornalista possa executar estas tarefas, independente de horário e de sua localização geográfica.

2.1 SISTEMAS DE GESTÃO DE CONTEÚDOS WEB

A gestão de conteúdo segundo Lapa (2004):

[1] http://www.cnn.com
[2] http://g1.globo.com
[3] http://www.folha.uol.com.br
[4] O webmaster é a pessoa que gerencia e mantém um site em funcionamento

> A gestão de conteúdo em *sites* pode ser vista como um conjunto de técnicas, definições e procedimentos de ordem estratégica e tecnológica visando integração de todos os processos relacionados à criação, agregação, personalização, entrega e arquivamento de conteúdos de uma organização. (LAPA, 2004, p. 40).

Com isso, é possível definir que um *Web Content Mangement Sistem* (WCMS) ou Sistemas de Gestão de Conteúdos Web (SGCW) são ferramentas que facilitam os usuários a criarem conteúdos, gerenciá-los e publicá-los simplificadamente sem que seja preciso ter contato com a programação do sitio.

Segundo Fraser (2002) para se ter bons resultados na implementação de um WCMS é necessário que o Gerenciador funcione como um *workflow*[5] e que trabalhe da seguinte forma: o jornalista cria o artigo, em seguida, ele é repassado para o editor para que seja revisado e corrigidos os erros, e é enviado ao publicador para que ele aprove e publique o artigo. A grande maioria dos WCMS na atualidade são estruturados neste padrão, o que facilita a migração entre os CMS.

Sistemas de Gestão de Conteúdos geralmente são projetados de maneira que forneçam funcionalidades básicas, sobre as quais se desenvolvem as aplicações mais próximas dos usuários finais (Pereira; Bax, 2002). Dentre essas funcionalidades destacam-se:

a) gestão de usuários (quem pode o quê e quando);

b) criação e armazenamento de conteúdos (permite os usuários adicionarem e editarem os conteúdos);

c) utilização de metadados[6] com a finalidade de facilitar a busca do conteúdo por sua descrição ou palavras chaves;

[5] Seqüência de passos necessários para que se possa atingir a automação de processos de negócio.
[6] Dados que descrevem outros dados. Metadados sobre um documento XML são definidos no DTD ou no documento XML propriamente dito.

d) controle de qualidade da informação, por meio do *workflow* descrito por Fraser (2002);

e) abstração de programação e interface com os usuários.

2.2 FERRAMENTAS DE GESTÃO DE CONTEÚDOS WEB

Existem no mercado inúmeras ferramentas em Software Livre escritas em diversas linguagens, como PHP, *Python*, ASP.NET, entre outras. Neste meio, algumas ferramentas vêm atingindo certo nível de popularidade, como é o caso das escritas em PHP como MAMBO, XOOPS, *PHP-Nuke*, *WordPress* e Joomla.

Para a realização desta pesquisa, foi utilizada a ferramenta CMS *Matrix*[7], a qual tem o objetivo de auxiliar o usuário (programador) na escolha de Gerenciadores de Conteúdos num processo de comparações entre diferentes CMS existentes no mercado. Esta ferramenta tem em seu banco de dados as características principais dos gerenciadores, tais como: requisitos, segurança, usabilidade, desempenho, manutenção, entre outros, como mostra a Tabela 1, onde são comparados os CMS: MAMBO, XOOPS, PHP-NUKE, *WordPress* e Joomla. Através deste comparativo, é possível perceber que a ferramenta que mais se destacou para a utilização deste trabalho foi o Joomla e a que menos supriu as necessidades foi o PHP-Nuke. Porém, isso é valido para o caso aqui estudado, não impedindo que a ferramenta Joomla seja melhor do que as concorrentes para outras finalidades. Por exemplo, caso o foco do trabalho seja uma ferramenta que facilite a criação de *blogs*, a melhor opção seria o *WordPress*, pois a mesma é especializada para este tipo de serviço. É necessário lembrar que deve-se levar em conta as datas de atualização das características de cada ferramenta, pois algumas,

[7] Portal CMS Matrix – http://www.cmsmatrix.org

como o Joomla, já estão em versões superiores e, consequentemente, sofreram algumas alterações.

Tabela 1: Comparativo entre CMSs

Características	**Joomla**	**Mambo Open Source**	**PHP-Nuke**	**WordPress**	**Xoops**
Ultima atualização	25/04/2006	16/12/2006	07/02/2006	25/07/2007	20/12/2006
Requisitos					
Servidor e Aplicação	Apache + PHP	Apache + PHP	Apache + PHP	Apache + PHP	Apache + PHP
Banco de Dados	MySQL	MySQL	MySQL, Postgres, mSQL, Interbase, Sybase	MySQL	MySQL
Sistema Operacional	Qualquer	Qualquer	Qualquer	Qualquer	Qualquer
Licença	GNU GPL	GNU GPL	GNU GPL	GNU GPL	GNU GPL
Segurança					
Aprovação de Conteúdos	Sim	Sim	Não	Sim	Sim
Autenticação LDAP	Plugin	Sim	Não	Não	Sim
Histórico de Login	Sim	Plugin	Não	Plugin	Plugin
Tratamento de Seções	Sim	Plugin	Não	Plugin	Limitado
Facilidades na Utilização					
Lista de discussão por E-mail	Plugin	Não	Não	Limitado	Não
URLs Amigáveis	Sim	Sim	Não	Sim	Plugin
Redimensionamento de Imagens	Sim	Não	Não	Limitado	Plugin
Prototipação	Sim	Não	Não	Não	Não
Linguagem de Template	Sim	Sim	Não	Não	Sim
WYSIWYG Editor	Sim	Sim	Não	Não	Sim
Desempenho					
Balanceamento de Carga	Não	Não	Não	Não	Sim
Armazenamento em *cachê*	Sim	Sim	Não	Plugin	Sim
Manutenção					
Escalonamento de Conteúdos	Sim	Sim	Não	Limitado	Sim
Administração *inline*	Sim	Sim	Não	Não	Sim
Administração *online*	Sim	Sim	Sim	Sim	Sim
Lixeira	Sim	Sim	Não	Não	Não
Estatísticas	Sim	Sim	Sim	Plugin	Plugin
Sub-Sites	Sim	Não	Nçai	Não	Plugin
Temas/Skins	Sim	Sim	Sim	Sim	Sim

Fonte: Adaptado de CMS Matrix (2007)

2.3 O CMS JOOMLA

O CMS Joomla não é simplesmente um Gerenciador de Conteúdos *Web*. O Joomla, juntamente com o conjunto de ferramentas compostas em sua API, se encaixa na definição de *framework*[8] de programação. Por meio dele, além de sua principal função de gerenciar arquivos de conteúdo, pode-se construir inúmeras aplicações complexas, tais como: gerenciadores de imagens, sistemas de tele-entrega, catálogo de produtos, entre outros. Sua arquitetura é sólida e, apesar de ser uma tecnologia recente é mantida por milhares de desenvolvedores espalhados por todo o mundo.

O Joomla facilita tanto à equipe de desenvolvimento quanto ao usuário. A equipe de desenvolvimento é favorecida pelo fato de não necessitar mais do trabalho seqüencial que existia no método tradicional. Com a nova metodologia de produção, pode-se desenvolver a programação em paralelo com o designer, proporcionando maior agilidade nos processos. O usuário é beneficiado, pois deixa de depender de um técnico ou de um programador para organizar e publicar os conteúdos dinamicamente.

Analisando o mercado, é possível perceber que muitas empresas, instituições e usuários confiaram no potencial do Joomla para suprir suas necessidades. Entre elas estão: Porsche Brasil, Conselho Nacional de Justiça (CNJ), Universidade Federal do Tocantins, Hospital de Clínicas de Porto Alegre, Ministério da Educação (MEC), Mitsubishi Venezuela e o site oficial do Michael Jackson.

[8] Um *framework* é um conjunto de componentes de software que provêem uma arquitetura e estrutura básica para o desenvolvimento de uma aplicação.

Figura 1: Site Ministério da Educação (MEC)
Fonte: Brasil (2007).

2.4 VERSÕES DO JOOMLA

A equipe de desenvolvimento do Joomla utilizou o *framework* do *Mambo Open Source* (MOS) para desenvolver as ferramentas desde a versão 1.0 até a atual 1.0.13. Porém, ainda existem muitas partes de códigos do antigo Mambo mescladas com o Joomla. Atualmente, o *Core Team* ou núcleo de desenvolvedores do projeto, está investindo largamente na nova versão 1.5 do *Joomla*, disponibilizada em versão experimental em 21 de Julho de 2007(Joomla, 2007). Este *framework* foi totalmente reescrito para que algumas funções possam ser inclusas nesta versão, o mesmo se torna cada vez mais distante do MOS, exceto por questões de compatibilidade de alguns aplicativos.

Neste trabalho será usada a versão 1.5 do *Joomla* pelos seguintes fatores:

a) separação da programação, lógica de negócios e interfaces dentro dos padrões *Model-View-Controller* (MVC);

b) administração compatível com muitas linguagens e paradigmas de programação, como o AJAX[9] e *Web Services*;

c) segue os padrões internacionais da *World Wide Web Consortion* (W3C);

d) utiliza do PHP5 ou superior, o que possibilita melhorias na Orientação a Objetos;

e) possui *File Transfer Protocol* (FTP) integrado na ferramenta;

f) possui a licença de Sofwtare Livre *Genal Public License*(GPL) e proíbe o fechamento de código para fins comerciais.

2.5 ESTRUTURA DO CMS JOOMLA

O CMS Joomla tem particularidades quanto a sua estrutura, o que facilita a organização dos arquivos e o desenvolvimento de aplicações. Ele é composto por *Front End* e *Back End*, níveis de acessos, conteúdos e extensões. A seguir faz-se um breve comentário sobre cada um dos itens:

a) *Front End*: é a parte do site que está visível aos usuários, onde ficam organizados os conteúdos e aplicações do site onde os usuários podem visualizar e interagir. O *Front End* é dividido em vários blocos (níveis) que facilitam sua customização, tais quais *Footer, Left, Inset, right, top, user1, user2*, entre outros, como mostra a Figura 2;

[9] Uso assíncrono de um combinado de tecnologias providas por navegadores Web, como javascript e XML, afim de tornar as paginas mais iterativas com o usuário.

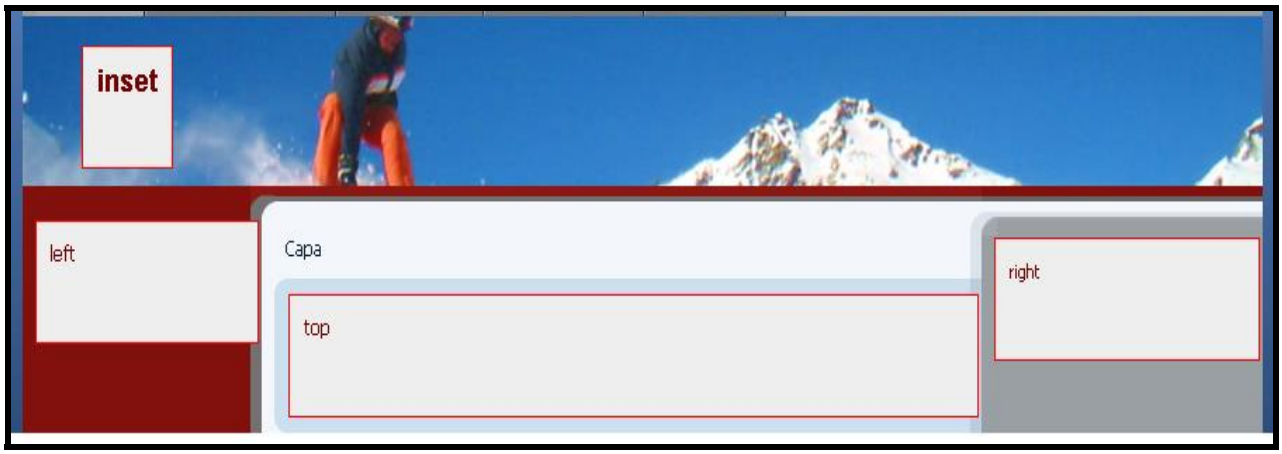

Figura 2: Separação do Front End
Fonte: JOOMLA (2008)

b) *Back End*: é a parte em que os desenvolvedores ou usuários podem fazer o gerenciamento do conteúdo propriamente dito. O mesmo conta com um painel de controle com inúmeras funções para que o usuário possa manter o site, ou até mesmo construí-lo usando as ferramentas dispostas no painel;

Figura 3: Back End do Joomla
Fonte: JOOMLA (2008)

c) níveis de acessos: são divididos em acessos ao *Front End* e acessos ao *Back End*. Os níveis de acessos do *Front End* são divididos em: (*i*) *registred*: usuário que pode visualizar conteúdos restritos no site; (*ii*) *Author*: *onde* usuário que pode submeter conteúdos ao site; (*iii*) *Editor*: usuário que pode submeter e editar conteúdos do site; e, (*iv*) *Publisher*: usuário que pode submeter, editar e publicar os conteúdos do site. Os níveis do *Back End* são classificados como: (*ii*) *manager*: onde o usuário pode administrar os conteúdos do site pelo *Back End*; (*ii*) *Administrator*: pode administrar os usuários, administrar conteúdos e modificar configurações; e, (*iii*) *Super administrator*: quem administra usuários, administra conteúdos, instala extensões, modifica configurações e gerencia os administradores;

d) gestão de conteúdos: A gestão de conteúdos é o princípio básico de um CMS. O que o Joomla faz é organizar o conteúdo que será exposto na *Web*, forçando o usuário a preocupar-se mais com a estrutura do conteúdo a ser exposto no site do que o resultado final (as páginas *web*), que é responsabilidade do *template* atribuído pelo designer do site ou portal. A organização dos conteúdos do Joomla facilita, não só a visualização e manutenção do site, mas também a busca por conteúdo em motores de sites como o Google[10] e Yahoo[11];

e) extensões: O Joomla, por ser um Gerenciador de Conteúdos, muitas vezes pode não suprir totalmente as necessidades de um site ou portal. Pensando nisso, os desenvolvedores criaram extensões que podem ser instaladas a partir do programa padrão, a fim de customizar o portal de acordo com as necessidades dos clientes. Estas são divididas em: componentes, módulos, *pug-ins* e *templates*.

[10] Google – http://www.google.com
[11] Portal Yahoo – http://www.yahoo.com

2.6 TRABALHOS CORRELATOS

Os trabalhos apresentados neste capítulo, em sua grande maioria, utilizam de técnicas de modelagem de dados UML e *Model Driven Architecture*[12] (MDA) para a automatização na geração de códigos fonte ou estruturas de controle.

A grande maioria dos trabalhos é voltada para a geração de códigos para frameworks de programação. Alguns, porém, abordam somente conceitos.Ambos vêem ao encontro deste projeto de pesquisa que tem como seu objetivo geral permitir a criação de componentes Joomla por intermédio de diagramas UML.

2.6.1 Ferramenta para a geração de códigos a partir da especialização do diagrama de classes

O objetivo do trabalho foi a criação de uma ferramenta que auxilia o processo de desenvolvimento de software que possibilita a especialização dos diagramas de classes UML. Esta ferramenta auxilia no processo de desenvolvimento de software por meio do enriquecimento das informações contidas em diagramas de classes UML (*Unified Modeling Language*) para geração de código. Ela foi desenvolvida utilizando a plataforma Microsoft. NET.

O processo de geração do código fora constituídos dos seguintes passos:

a) gerar os diagramas de classes por meio da ferramenta case *Enterprise Architect*;

b) criar as estruturas básicas e de especialização com o objetivo de receber o arquivo XMI provido pela ferramenta case UML;

[12] A MDA é um padrão definido pela OMG que tem como principal iniciativa a geração de código fonte por meio de modelagem do software.

c) exportar os arquivos XMI por meio da ferramenta *Enterprise Architect*;

d) importar o arquivo de modelos XMI;

e) gerar o código por meio de um *plug-in*.

As conclusões realizadas pelo autor mostraram a viabilidade de sua ferramenta que provê melhorias na qualidade do software desenvolvido e redução do tempo de desenvolvimento. O autor também destaca pontos importantes, como a utilização de *plug-ins* e suporte a múltiplos idiomas, o que torna sua ferramenta flexível.

2.6.2 Uma ferramenta para aprendizagem a desenvolvimento de sistemas

O trabalho desenvolvido por Gonçalves (2004), propõe auxiliar o processo de aprendizagem sobre desenvolvimento de sistemas computacionais. Para isso, foi criada uma ferramenta que permite a modelagem visual de diagramas de classes UML e, a partir deste, possibilite a geração automatizada de código.

A pesquisa se divide em duas partes; a primeira foca no desenvolvimento de uma ferramenta “Modeladora de Sistemas”. Essa tem a capacidade de oferecer subsídios para que os usuários consigam, por meio da interface visual e das ferramentas utilizadas, a criação de diagramas de classes UML. Assim, pode ser realizada a geração do arquivo XMI onde está contido as informações do modelo. A segunda parte do trabalho se destina à criação de um “Gerador de Sistemas”, que tem por finalidade gerar o código básico a partir do modelo (classes, *getters* e *setters*), gerar uma camada de persistência de dados e criar interfaces visuais para a inclusão, remoção e alteração de entidades do sistema.

A linguagem que foi adotada para o desenvolvimento do projeto é o *Smalltalk*, pois, segundo o autor, a mesma é de fácil aprendizado e utilização. O autor

conclui conseguiu-se alcançar o objetivo almejado, que era uma ferramenta de modelagem que auxiliasse a aprendizagem de sistemas computacionais e com suporte a persistência de dados.

2.6.3 Modelação de workflows e ferramentas declarativas

O trabalho desenvolvido por Gamito, Cunha e Abreu (2007) consiste na modelagem de *workflows* por meio de diagramas de atividades e por meio do desenvolvimento de uma ferramenta que proporcione a geração de códigos para o *framework* ISCO, que tem por finalidade fazer o controle de aplicações para *workflow*.

O objetivo do trabalho inicia na modelagem do diagrama de atividades, esse feito por meio da ferramenta case ArgoUML, eleita por vários fatores, incluindo licença, flexibilidade e portabilidade. Logo após os autores se propõem a desenvolver um aplicativo que faça o mapeamento do arquivo XMI, que contém as informações de modelagem dos diagramas e, a partir deste, gerar o código para a plataforma ISCO.

Os autores concluem que foram obtidos resultados satisfatórios, pois a ferramenta apresenta diversas vantagens quanto a sua utilização, uma vez que ela facilita a compreensão do funcionamento dos sistemas por meio dos diagramas de classes UML. Por meio dela, é possível também se obter acesso direto ao motor de execução dos *workflows*.

2.6.4 ODYSSEY-MDA: Uma ferramenta para transformações de modelos UML

O trabalho desenvolvido por Maia, Blois e Werner (2005) apresenta a ferramenta Odyssey-MDA para a realização de transformações bidirecionais sobre

modelos UML. A ferramenta faz a transformação de modelos UML para códigos de programação, possibilitando o desenvolvimento de *plug-ins* para a geração de códigos de diversas plataformas de desenvolvimento.

A arquitetura da ferramenta consiste em três fases:

a) o mapeamento do arquivo XMI, provido por ferramentas CASE UML;

b) a interpretação e mapeamento do arquivo de informações dos modelos XMI;

c) a geração de códigos por meio de técnicas de MDA.

A Odyssey-MDA foi desenvolvida em JAVA e está disponível em versão de uso acadêmico no site dos desenvolvedores (MAIA, 2007) sua interface é mostrada na Figura 4.

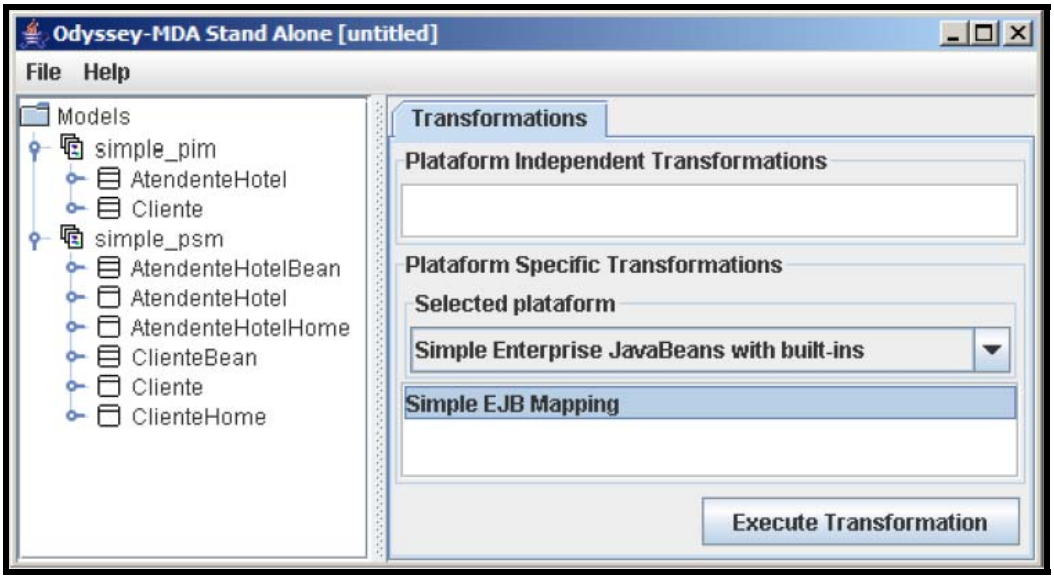

Figura 4: Interface do Odyssey-MDA.
Fonte: MAIA (2007).

Os autores concluem que atualmente a Odyssey-MDA trabalha somente com UML, mas futuramente os mesmos pretendem expandí-la fazendo com que seja capaz de transformar qualquer modelo que seja baseado no *Managed Object Format* (MOF).

3 ENGENHARIA DE SOFTWARE

Uma empresa bem sucedida não é aquela que somente utiliza as últimas tecnologias do mercado, possui ótima documentação e têm funcionários qualificados. Empresas que não planejam suas atividades correm sérios riscos quanto ao seu sucesso. Segundo Diniz (2004), para uma pessoa ter sucesso na vida e nos negócios “é necessário planejar”. As chances de uma pessoa que põe seus objetivos em pauta ter sucesso são muito maiores em relação as que não se planejam.

No caso de *Software Houses*, o mais importante para o sucesso da mesma é produzir software de qualidade, atendendo as necessidades dos seus usuários e mantendo prazos previsíveis para a conclusão do projeto. Para que isto ocorra, é imprescindível ter um planejamento e modelagem do projeto de software.

3.1 MODELAGEM DE SOFTWARE

Existem limites para a capacidade humana compreender complexidades. Por exemplo, com pedaços de madeira, alguns pregos e telhas, é possível realizar a construção de uma casa para um cão que atenda as necessidades do animal sem que seja preciso a elaboração de um projeto engenhoso para isto. Entretanto, o caso não se aplica para a construção de um prédio ou um edifício empresarial. Possivelmente, pode-se criar uma pequena construção de maneira que atenda as necessidades de uma pequena empresa sem um projeto arquitetônico. Porém, à medida que a empresa cresce, surge a necessidade de ampliar suas estruturas. A ausência de uma planta da construção dificultaria muito o trabalho dos construtores que provavelmente optariam por

abandonar a antiga estrutura física e construir uma nova estrutura utilizando um projeto de engenharia (BOOCH; RUMBAUGH; JACOBSON, 2000).

Na criação de um *software* o processo é semelhante. É possível desenvolver um sistema sem um projeto de modelagem. Porém, esse sistema falharia à medida que o mesmo necessite de customizações, adaptações, reparos ou até mesmo mudanças em sua equipe de desenvolvimento.

De acordo com Booch, Rumbaugh e Jacobson (2000, p. 3), “para desenvolver software de qualidade duradoura, será necessário criar uma arquitetura de fundação sólida que aceite modificações”.

Os modelos fornecem uma cópia de como será o sistema após sua conclusão. Tais modelos podem abordar uma forma mais detalhada de como será o sistema, assim como pode generalizá-lo, mostrando uma visão mais panorâmica.

Ainda segundo Booch, Rumbaugh e Jacobson (2000), com a modelagem de software pode-se alcançar quatro objetivos principais:

a) visualizar como o sistema é ou como se deseja que seja;

b) permitir especificação da estrutura ou comportamento de um sistema;

c) proporcionar um guia para a construção do sistema;

d) documentar as decisões tomadas.

Geralmente, os modelos, tanto na construção civil quanto no desenvolvimento de software, são representados graficamente por meio de diagramas, devido à capacidade que o ser humano tem de assimilar as imagens mais rapidamente ao cérebro do que qualquer outro meio, como a escrita e a fala. (FONSECA, 1985). No entanto, os modelos também contemplam informações textuais, cujo objetivo é a explanação ou definição de certas partes do diagrama. A representação gráfica unida com as informações textuais forma a documentação do modelo (BEZERRA, 2002).

3.2 ORIENTAÇÃO A OBJETOS

O paradigma da Orientação a Objetos (OO) formulou-se a partir da chamada “Analogia biológica” de Alan Kay (BEZERRA, 2002) que foi um dos pais da OO. Kay imaginava como seria um sistema de software que se comportasse como um ser vivo. Neste sistema, cada parte (chamada de célula) se comunicaria com outras células por meio do envio de estímulos para realizar um objetivo comum, sendo que cada célula se comportaria de forma independente.

A partir desta teoria Alan Kay definiu os seguintes princípios da Orientação a Objetos:

a) qualquer coisa é um objeto;

b) objetos realizam tarefas por meio de requisição de serviços a outros objetos;

c) cada objeto pertence a uma determinada classe;

d) uma classe agrupa objetos similares;

e) classes são organizadas em hierarquias.

Há dois fatores que devem ser levados em consideração no desenvolvimento orientado a objetos. *(i)* a OO é uma linguagem próxima entre analistas, desenvolvedores e usuários por ser focada em objetos do mundo real; *(ii)* a possibilidade de reaproveitamento de objetos, reduzindo assim, tempo de produção, custos, entre outros (BOOCH, RUMBAUGH, JACOBSON, 2000).

3.2.1 Classes e Objetos

A partir dos princípios da OO pode-se definir que tudo é um objeto, ou seja, Rodrigo, cliente, fornecedor seriam exemplos de objetos (WELLING; THOMSON, 2003).

Tomando como exemplo o objeto c*liente X*, este objeto tem várias características como nome, telefone, e-mail. Essas características são comuns a objetos como c*liente Y* e c*liente Z* e poderão ser usadas em comunhão.

O agrupamento das características dos objetos que se assemelham é chamado de classes. Uma classe é um modelo (*template*) que possibilita a criação de novos objetos. Ao instanciar um objeto de uma classe, define-se que as características agrupadas na classe farão parte do objeto criado (BOOCH; RUMBAUGH; JACOBSON, 2000).

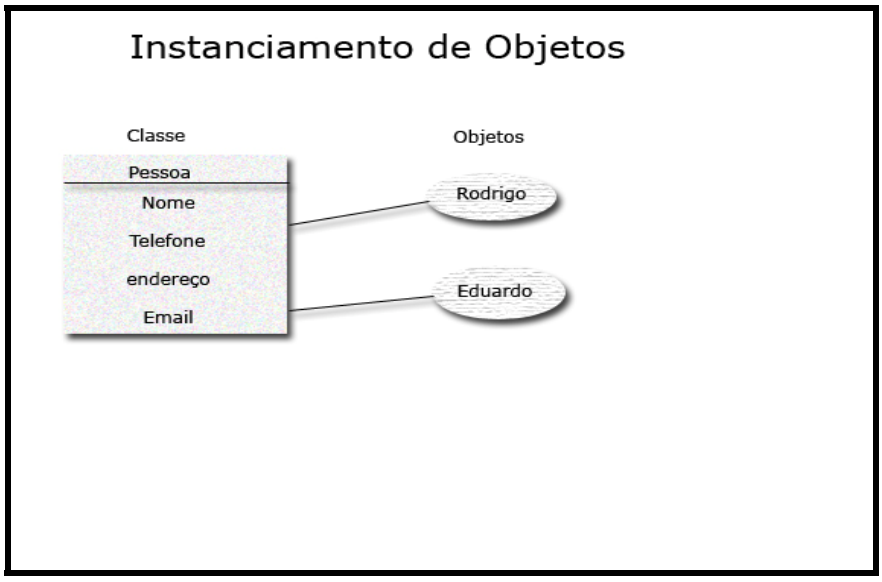

Figura 5: Instancia de Objetos

3.2.2 Estímulos

Os objetos se relacionam entre si enviando estímulos (mensagens) uns aos outros. Um estímulo é um evento que permite que os objetos se comuniquem (CONVERSE; PARK; MORGAN, 2004).

Um objeto, ao reconhecer um estímulo gerado de outro objeto imediatamente invocará algum serviço para atender ao pedido daquele que originou o estímulo. Por exemplo, quando o objeto humano recebe o estímulo pular, ele interpreta-o e realiza as operações que já foram pré-definidas para ele como mover pernas e braços como vistos na Figura 6.

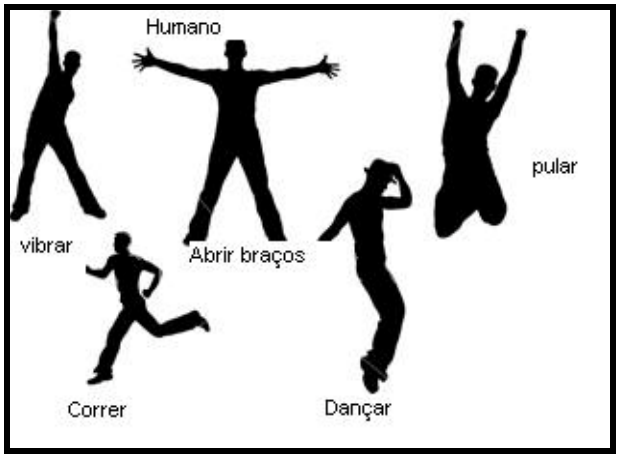

Figura 6: Estímulos

3.2.3 Encapsulamento

Todas as informações dentro de um sistema OO são armazenadas nos seus objetos e podem ser manipuladas somente quando um objeto é acionado para realizar ou executar uma operação ou serviço (CONVERSE; PARK; MORGAN, 2004).

O comportamento e a informação são encapsulados no objeto. A única forma de afetar o objeto é executar operações dele para si mesmo. Essa característica é também conhecida por *information hiding*, que significa esconder a estrutura interna (CONVERSE; PARK; MORGAN, 2004).

Para executar um objeto não é necessário saber seu comportamento ou como suas informações são representadas, somente é preciso saber quais as operações que o mesmo oferece. Dessa forma, o termo encapsulamento (Figura 7) significa que tudo o que se consegue ver em um objeto é sua interface, nomeada pelas operações que podemos executar sobre o objeto.

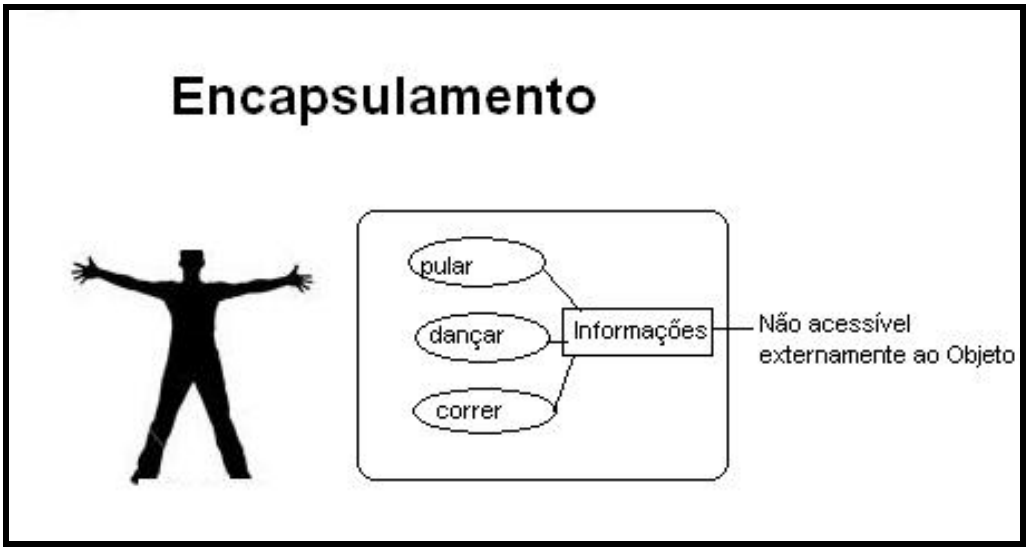

Figura 7: Encapsulamento

3.2.4 Abstração

O conceito de abstração é uma importante característica do paradigma OO, por meio dela pode-se observar os objetos com um foco bem definido. Desta forma, podemos dizer que abstração consiste em desprezar algumas informações que estão presente na realidade que se está modelando, porém não relevantes para o modelo que

se pretende construir relativo à perspectiva do observador (CONVERSE; PARK; MORGAN, 2004).

3.2.5 Herança

A herança tem como objetivo criar um relacionamento hierárquico entre as classes. Ela compreende *superclasses* que geram características e *subclasses* que herdam essas características das classes geradoras (WELLING; THOMSON, 2003).

Assim, pode-se ter uma superclasse chamada DadosPF cujo conteúdo compreende os dados comuns de uma pessoa física (nome, cpf, idade, sexo...) e subclasses que herdam essas características, como ClientePF, e implemente outras características como codigoCliente, faturamentoMensal, dataAquisicao, entre outras (DEITTEL; DEITTEL, 2003).

3.2.6 Polimorfismo

O conceito de polimorfismo tem o significado de que diferentes classes podem ter comportamentos distintos para a mesma operação (WELLING; THOMSON, 2003). Com o polimorfismo, é possível desenvolver códigos de programação facilmente extensíveis. Os sistemas podem ser escritos para processar as informações de uma maneira genérica, incentivando assim, a reutilização de códigos tornando o sistema otimizado (DEITTEL; DEITTEL, 2003).

3.3 UNIFIED MODELING LANGUAGE (UML)

Segundo os criadores da linguagem, Grady Booch, James Rumbaugh e Ivar Jacobson, “a *Unified Modeling Language* (UML) é uma linguagem padrão para a elaboração da estrutura de projetos de Software” (BOOCH; RUMBAUGH; JACOBSON, 2000) que foi reconhecida pela OMG em 1997, e desde então tem tido uma grande aceitação pela comunidade de desenvolvedores de sistemas.

A visão geral da UML pode ser empregada em diversas áreas do desenvolvimento de software com finalidades de auxílio a visualização, especificação, construção e documentação de sistemas de software com o intuito de abstrair e gerenciar a complexidade do mesmo.

Muitos programadores inexperientes constroem sistemas sem utilizar modelos para os mesmos. Entretanto, este processo de não se ter um padrão, pode gerar vários problemas. A comunicação entre a equipe de desenvolvimento e o profissional que gerou os modelos está sujeita à falhas, pois dificulta o entendimento de consultores externos e de profissionais iniciantes.

A partir de um sistema visual fica muito mais simples o entendimento de, por exemplo, uma hierarquia de classes, que, ao ler o código, seu significado pode ser inferido, mas não percebidos diretamente, de maneira semelhante a exportações de objetos à *web*. Também, uma grave falha de não se ter um modelo visual do projeto é a remoção do código, uma vez que o desenvolvedor retire uma parte do código do projeto, este automaticamente é perdido e impossibilitado de ser usado para programas futuros (BOOCH; RUMBAUGH; JACOBSON, 2000).

Porém, a UML não está restrita na linguagem visual e ao auxílio a construção de softwares e diagramação. A UML contém muitos artefatos que podem ser utilizados como documentação do software, incluindo análise de requisitos, arquitetura, projeto, código fonte, planos de projetos, testes, protótipos e versões.

"*A UML é independente de linguagens de programação quanto aos processos de desenvolvimento*" (BEZERRA, 2002). Isto quer dizer que a UML não se limita a uma linguagem de programação e nem a maneira de desenvolvimento adotada. Isto é muito importante, pois sistemas de softwares distintos requerem diferentes formas de abordar problemas.

Apesar de a UML abordar o conceito de Orientação a Objetos, a mesma obteve sucesso em diversas áreas, não se limitando ao desenvolvimento de software, como por exemplo, sistemas de vendas, transportes, defesa do espaço aéreo, entre outras (BOOCH; RUMBAUGH; JACOBSON, 2000).

3.3.1 Diagramas UML

Um sistema de software complexo necessita que seus desenvolvedores tenham a capacidade de analisar o modelo de diversos aspectos. Para que essa compreensão fosse facilitada foram desenvolvidos os diagramas.

Um diagrama pode ser definido por ser uma apresentação gráfica de elementos, geralmente representados por gráficos e relacionamentos (BOOCH; RUMBAUGH; JACOBSON, 2000).

A UML inclui nove diagramas, estes:

a) diagrama de classes[13];

b) diagrama de objetos[14];

c) diagrama de casos de uso[15];

d) diagrama de seqüência[16];

[13] Representa um conjunto de classes, interfaces e colaborações e seus relacionamentos.
[14] Mostra um conjunto de objetos e seus relacionamentos.
[15] Organiza os comportamentos (parte funcional) do sistema.
[16] Tem como foco a ordem temporal das mensagens.

e) diagrama de colaborações[17];

f) diagrama de estados[18];

g) diagrama de atividades[19];

h) diagrama de componentes[20];

i) diagrama de implantação[21];

Apesar de todos serem muito bem empregados em diversas áreas do desenvolvimento de software, este trabalho será limitado ao estudo do diagrama de classes.

3.3.2 Diagramas de Classes

É muito comum encontrar diagramas de classes em projetos de software, por menores que sejam, pois eles compreendem toda a estrutura da programação do projeto em uma vista panorâmica e didática.

O diagrama de classes é composto por:

a) Conjunto de classes: vistas no item 4.2.1;

b) Interfaces: essas são uma coleção de operações utilizadas para visualizar, especificar, construir e documentar a associação interna do sistema;

c) Colaborações: uma colaboração é um aglomerado de classes, interfaces e outros elementos que trabalham em conjunto de forma cooperativa maior do que a soma de todas as suas partes;

[17] Mostra a organização estrutural dos objetos que enviam e recebem mensagens.
[18] Os estados de mudanças de um sistema orientado por eventos.
[19] Fluxo de controle de uma atividade para a outra.
[20] Conjunto de componentes e seus relacionamentos.
[21] Mostra o conjunto de nós e seus relacionamentos.

d) Relacionamentos de dependências, generalizações e associações: são as interconexões dos.

A Figura 8 mostra um exemplo de um diagrama de classes e seus componentes.

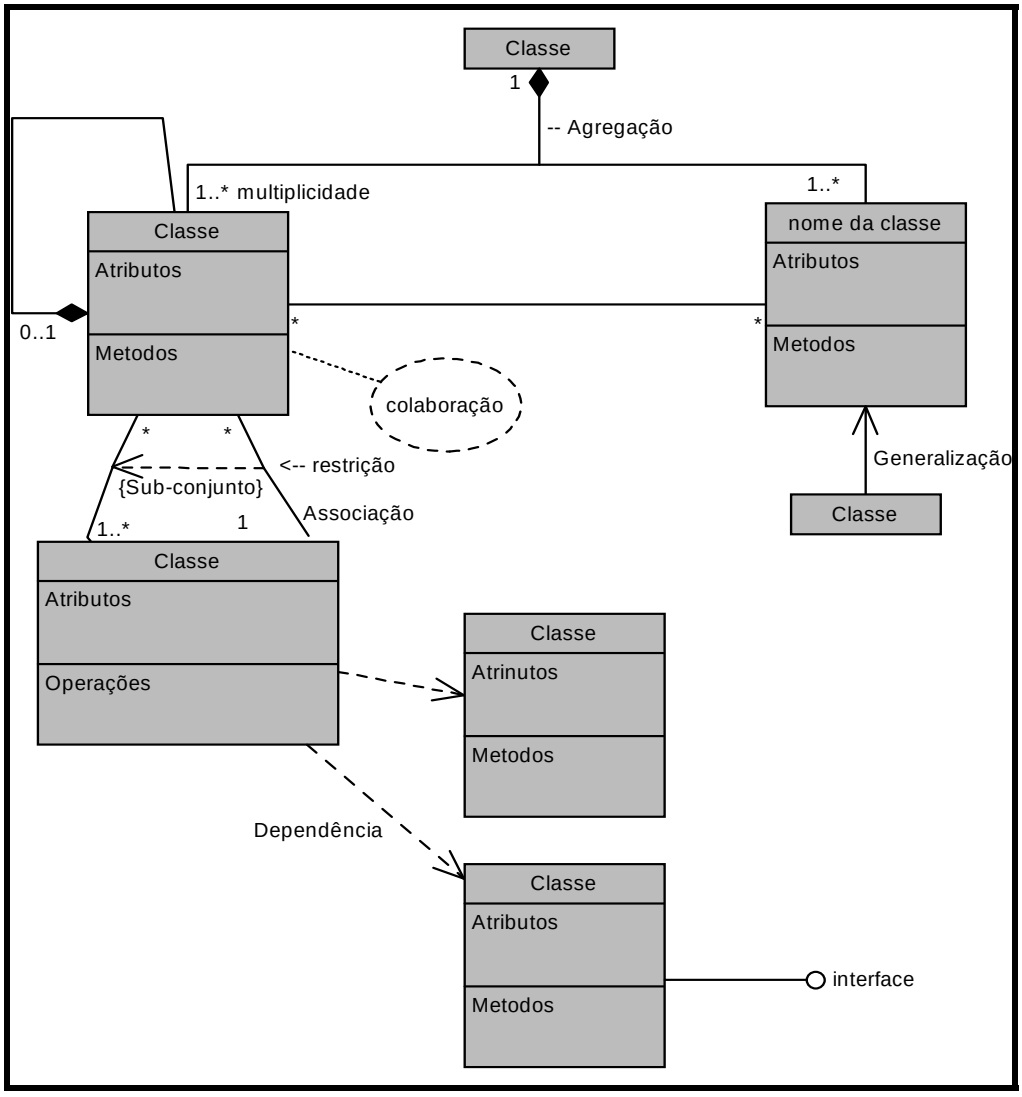

Figura 8: Diagrama de Classes e seus componentes

Os diagramas de classes também servem como base para outros diagramas, como os de componentes e de implantação (BOOCH; RUMBAUGH; JACOBSON, 2000). Tais representações gráficas têm por finalidade criar uma visão sistêmica,

especificar e documentar projetos. Além disso, fornecem subsídios para a construção de sistemas por intermédio da engenharia reversa (BEZERRA, 2002).

3.4 CODIFICAÇÃO DE DIAGRAMAS DE CLASSES UML

Existem no mercado padrões para a codificação dos arquivos de saídas originados pelas ferramentas case UML. O mais comumente encontrado nas ferramentas é o *XML Metadata Interchange* (XMI), definido pela OMG como padrão de troca de informações entre ferramentas UML. O XMI é baseado no XML e será abordado nos tópicos seguintes.

3.4.1 XML

A *Extensible Markup Language* (XML) é uma linguagem de definição para linguagens de marcação, especificada pela W3C para uso especialmente via *web*, contudo não se limita a isso (HOLZNER, 2001).

Ela também pode ser utilizada para a troca de informações entre aplicativos e sistemas independentes de sua plataforma, parametrização e configuração de aplicativos, padronização de técnicas como o RSS e *webservices*[22], entre outros.

Dentre tantas características presentes no XML é importante ressaltar que:

a) as informações que são passadas por meio desta tecnologia são prescritas dentro de *tags* em um documento de texto, normalmente chamado de instância XML;

[22] são componentes que permitem às aplicações enviar e receber dados em formato XML . Cada aplicação pode ter a sua própria “linguagem”, que é traduzida para uma linguagem universal, o formato XML (HOLZNER, 2001) .

b) o XML define regras para sua estrutura, que são especificadas dentro de um arquivo DTD[23], essas regras geralmente são chamadas de gramática e possui extensão DTD;

c) diferente do HTML, no XML é possível a criação de *tags*, como por exemplo, caso seja necessário especificar um endereço dentro do arquivo, poderá ser criado as tags *<endereço>Rua ...</endereço>*;

d) o XML é estruturado em uma hierarquia baseada em árvore, como por exemplo:

<pessoa>

<nome>Rodrigo</nome>

<sexo>Masculino</sexo>

<altura>1,70</altura>

</pessoa>

Desta forma, é possível a navegação pelas estruturas da árvore;

e) por meio das regras definidas pelas DTDs, é possível fazer uma validação dos dados contidos nos documentos XML. A tecnologia responsável por esta tarefa chama-se esquemas, e rigorosamente analisa o documento XML em busca de más formações. E, caso ache algum erro, a operação é abortada (PENDER, 2003).

No tópico seguinte será visto uma extensão da XML, a XML *Metadata Interchange* (XMI) e suas aplicações.

[23] contém regras que definem quais as tags que podem ser usadas em um documento XML e quais os valores válidos (MENDES, 2004).

3.4.2 XMI

A XML *Metadata Interchange* (XMI) é uma especificação criada pela OMG que visa facilitar o intercâmbio de informações entre as ferramentas de modelagem UML. Por meio desta, é possível que desenvolvedores terceiros possam utilizar-se dos *metadados* contidos nos *metamodelos* UML para a geração de códigos tais como Java, Visual Basic e PHP (HAROLD, 2004). A última é o foco da presente pesquisa

As informações que são trocadas por meio da XMI são compostas de dois artefatos básicos: as informações do modelo UML e os diagramas.

As informações do modelo UML compreendem a estrutura de pacotes dos modelos e todas as informações decorrentes do modelo como: classes, heranças, conexões, atributos, métodos, atores, objetos, dependências e assim por diante.

A informação diagramática consiste nas informações específicas do diagrama como: posição dos diagramas (coordenadas x e y), cores, tamanhos de fontes, símbolos, formas exatas, entre outros.

A XMI inclui:

a) XML *Document Production Rules*: regras utilizadas para codificar os diagramas das ferramentas UML e transformar em XMI;
b) XML *Document Type Definition* (DTD) *Production Rules*: regras usadas para a produção do documento, semelhante a descrita no tópico sobre XML;
c) XML *Schema Production Rules*: utilizando essas regras, as ferramentas podem validar a exatidão do documento XMI.

Existem dois níveis de exatidão do documento instância XMI: o documento bem formado e o documento válido. Define-se um documento bem formado, se o

mesmo contiver dois ou mais elementos e esses estiverem associados a um elemento raiz e todos os elementos estejam aninhados corretamente. Um documento válido é o documento cujo todos seus elementos sigam corretamente as regras expostas em sua DTDs e *Schemmas*.

Atualmente, existem muitas variações da XMI, cada qual com suas peculiaridades, ou seja, uma ferramenta que contemple seu sistema com a XMI versão 1.4 não se comunica com outra que utilize a versão 1.1. As versões mais comuns encontradas no mercado são a XMI 1.0 e a XMI 1.1.

A estrutura de um documento XMI compreende um cabeçalho, onde são definidos as informações dos componentes, e uma seção de conteúdo que contém todas as informações contidas acerca do modelo UML.

No próximo tópico é abordada a técnica utilizada para a geração de sistemas utilizando-se das informações extraídas do arquivo XMI.

3.5 ARQUITETURA ORIENTADA POR MODELOS

A capacidade humana de compreender complexidades é limitada (BOOCH; RUMBAUGH; JACOBSON, 2000). Por este motivo, diferentes técnicas são criadas para auxiliar em tarefas como desenvolvimento de software. Dentre essas técnicas, podemos destacar a Orientação a Objetos, UML, MVC, e *Model Driven Architecture* (MDA).

A Arquitetura Orientada por Modelos (MDA) foi uma iniciativa do grupo de gerenciamento de Objetos (OMG). Eles conceberam um padrão tendo como base a modelagem, seguindo a idéia de que a arquitetura de um projeto de software é a melhor base para se desenvolver sistemas e dar manutenção aos mesmos. (MELLOR et al.,

2005). O padrão consiste em atribuir tecnologias para que torne possível a geração de aplicações por meio de metamodelos

A MDA, além de estabelecer um padrão com o intuito de manter a interoperabilidade entre diferentes plataformas, oferece um conjunto de tecnologias e técnicas associadas para permitir que isso seja possível, tais como:

a) UML: a mais popular entre as tecnologias, é usada para capturar os modelos de semântica abstrata, bem como a estrutura de software de sistemas orientados a objetos;

b) MOF: um metamodelo em uma linguagem para descrever metamodelos.

Apesar de a MDA ser um padrão reconhecido pela OMG, algumas tecnologias precisam ser desenvolvidas enquanto outras ainda precisam de definição (MELLOR et al., 2005).

3.5.1 Vantagens no desenvolvimento por MDA

A seguir são abordadas algumas vantagens da utilização de uma aplicação gerada pelo processo de MDA:a

a) Aumentar a produtividade do programador em conseqüência da geração automatizada do código das aplicações;

b) Melhorar a consistência do código e da manutenção, pois permite o uso dos padrões de desenvolvimento e engenharia de software, proporcionando aos desenvolvedores e engenheiros de software um melhor entendimento da aplicação que está em desenvolvimento;

c) Otimizar o código fonte por meio de reutilização de códigos programados nos plug-ins;

d) Automatizar a portabilidade para outras plataformas por meio de plug-ins;

e) Utilização de padrões de projetos (declarações formais de semântica) para a confecção do código fonte, aumentando a qualidade e o tempo de vida útil do projeto.

f) Entre outras.

No próximo capítulo será explicado como foram aplicados todos os conhecimentos explanados até então.

4 A FERRAMENTA DESENVOLVIDA

A princípio, o Joomla foi criado para administrar conteúdos e artigos, porém, é possível desenvolver inúmeras aplicações complexas por meio dele. Carrinhos de compras, fóruns, perfis de usuários, listagem de produtos, são exemplos de casos em que podem ser incorporados ao Joomla.

O Joomla fornece vasta portabilidade para diversos sistemas e linguagens de programação. Seu núcleo é escrito em PHP e utiliza o MySQL como banco de dados, mas pode ser mesclado com outras linguagens, como é o caso do *JavaScript*, A*ctionScript*, entre outros.

Dentre todas as extensões disponíveis para Joomla, as mais importantes são os componentes. Por meio deles é possível ter acesso ao banco de dados e disponibilizar as informações aos usuários.

A ferramente desenvolvida neste trabalho foi batizada de JoomGen. Ela tem por finalidade extrair as informações da modelagem visual dos diagramas de classes UML e, por meio desses, proporcionar a geração automática da estrutura de um componente Joomla, como ilustra a Figura 9.

O Joomgen é um software livre sob licença General Public Licence (GPL), ou seja, seu código é aberto e pode ser modificado e distribuído livremente, desde que sejam mantidos os direitos autorais.

O sistema operacional utilizado para seu desenvolvimento foi o MAC OSX 10.4, que é multiplataforma, porém, seu funcionamento foi testado apenas em ambiente MAC OSX e Linux.

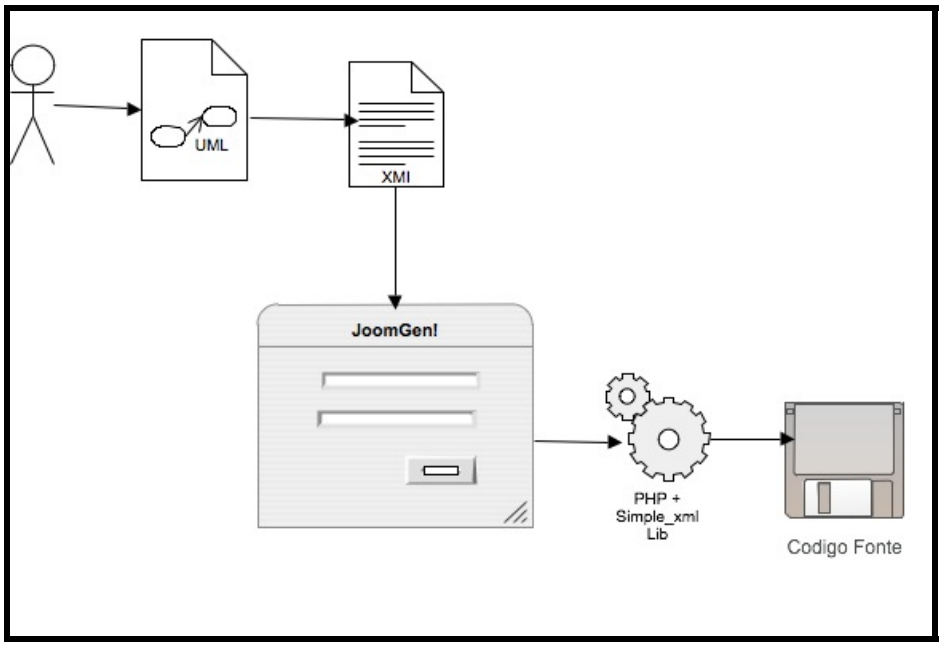

Figura 9: Funcionamento da ferramenta JoomGen

O funcionamento do JoomGen segue os seguintes passos:

a) o programador ou o engenheiro de software constrói o diagrama de classes do componente que pretende ser criado na ferramenta de UML de sua preferência[24] e o exporta para o formato XMI;

b) em seguida, importa-se o arquivo de saída correspondente ao diagrama na ferramenta desenvolvida, JoomGen;

c) o JoomGen extrai os dados do arquivo XMI e transforma-os em código fonte adaptado ao framework do Joomla e disponibiliza este por meio de um arquivo compactado;

Desta forma, o programador é abstraído de toda a parte burocrática do Joomla a qual se destina à padronização dos componentes.

[24] Para isto, é necessário que a ferramenta suporte a exportação dos diagramas no formato XMI 1.3.

4.1 METODOLOGIA

Para o desenvolvimento do trabalho, inicialmente foram definidos o tema e o escopo do trabalho junto ao orientador, seguido pela elaboração da proposta do projeto de pesquisa. Uma vez aceita a proposta, iniciou-se o levantamento bibliográfico dos temas abordados no trabalho. Na etapa seguinte, foi definida a linguagem de programação e a ferramenta case UML. Foram também realizados testes com versões de arquivos XMI e manipulação de arquivos XML com PHP. Definido o ambiente de programação, deu-se inicio à modelagem da ferramenta, programação e testes com os arquivos gerados pela mesma. Na etapa seguinte, foram realizados experimentos com códigos gerados pela ferramenta no ambiente Joomla. O último passo foi a documentação da ferramenta desenvolvida.

4.2 MODELAGEM

Segundo os conceitos apresentados no capítulo 3, percebe-se o quão importante é a modelagem de um sistema antes de sua construção propriamente dita.

Os 3 itens a seguir utilizam a linguagem UML para mostrar, em uma abordagem gráfica, o funcionamento da ferramenta desenvolvida para o propósito deste trabalho.

4.2.1 Diagrama de Classes

Este diagrama dentre outras funcionalidades mostra as classes, atributos, métodos, relacionamentos e interfaces da ferramenta JoomGen.

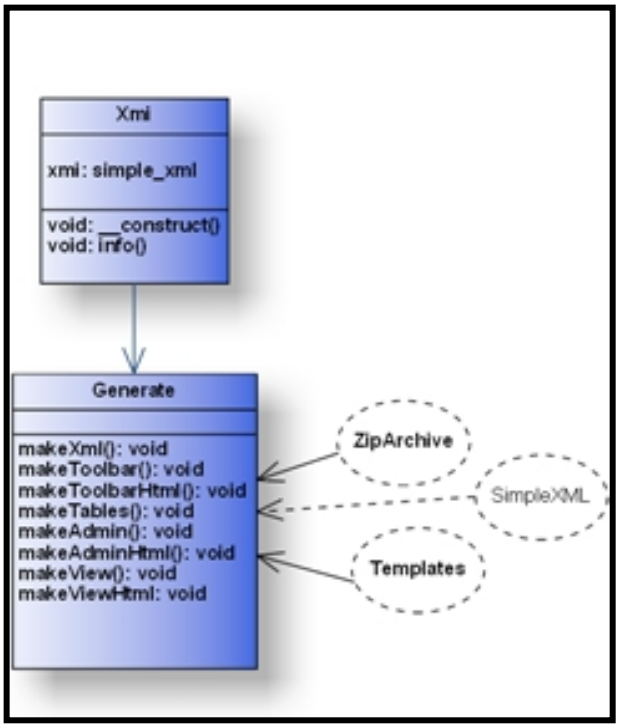

Figura 10: Diagrama de Classes

O JoomGen é composto por duas classes, a XML e a Generate. A primeira faz a leitura do arquivo proveniente do arquivo da modelagem UML e a segunda, uma classe filha, que faz a geração dos arquivos. Esta também conta com funções externas de compactação de arquivo, manipulação de XML e os arquivos de *templates*, um para cada tipo de arquivo gerado.

4.2.2 Diagrama de Componentes

Este diagrama tem por finalidade mostrar os componentes físicos do sistema, como clientes, servidores, bancos de dados, entre outros.

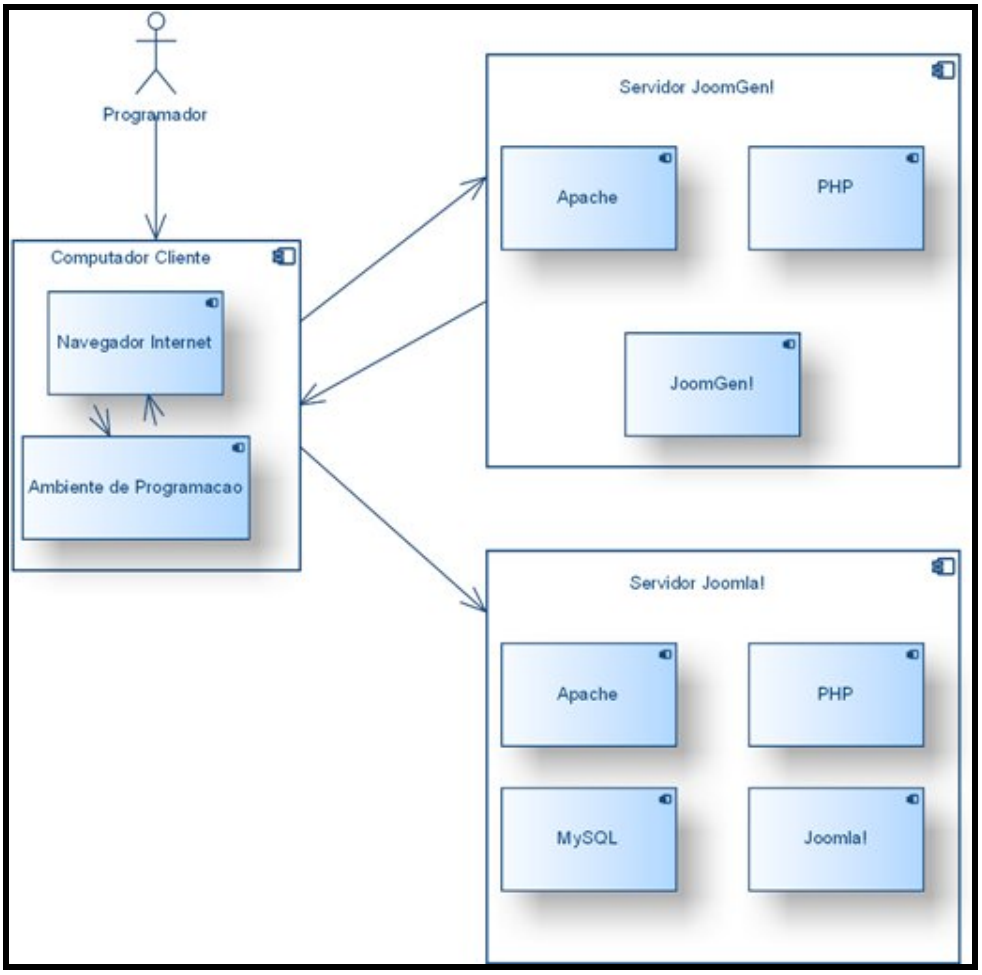

Figura 11: Diagrama de Componentes

4.2.3 Diagrama de Atividades

O diagrama de atividades mostra o fluxo de controle de um sistema. Normalmente, demonstra um processo seqüencial de uma atividade para a outra. (BOOCH; RUMBAUGH; JACOBSON, 2000).

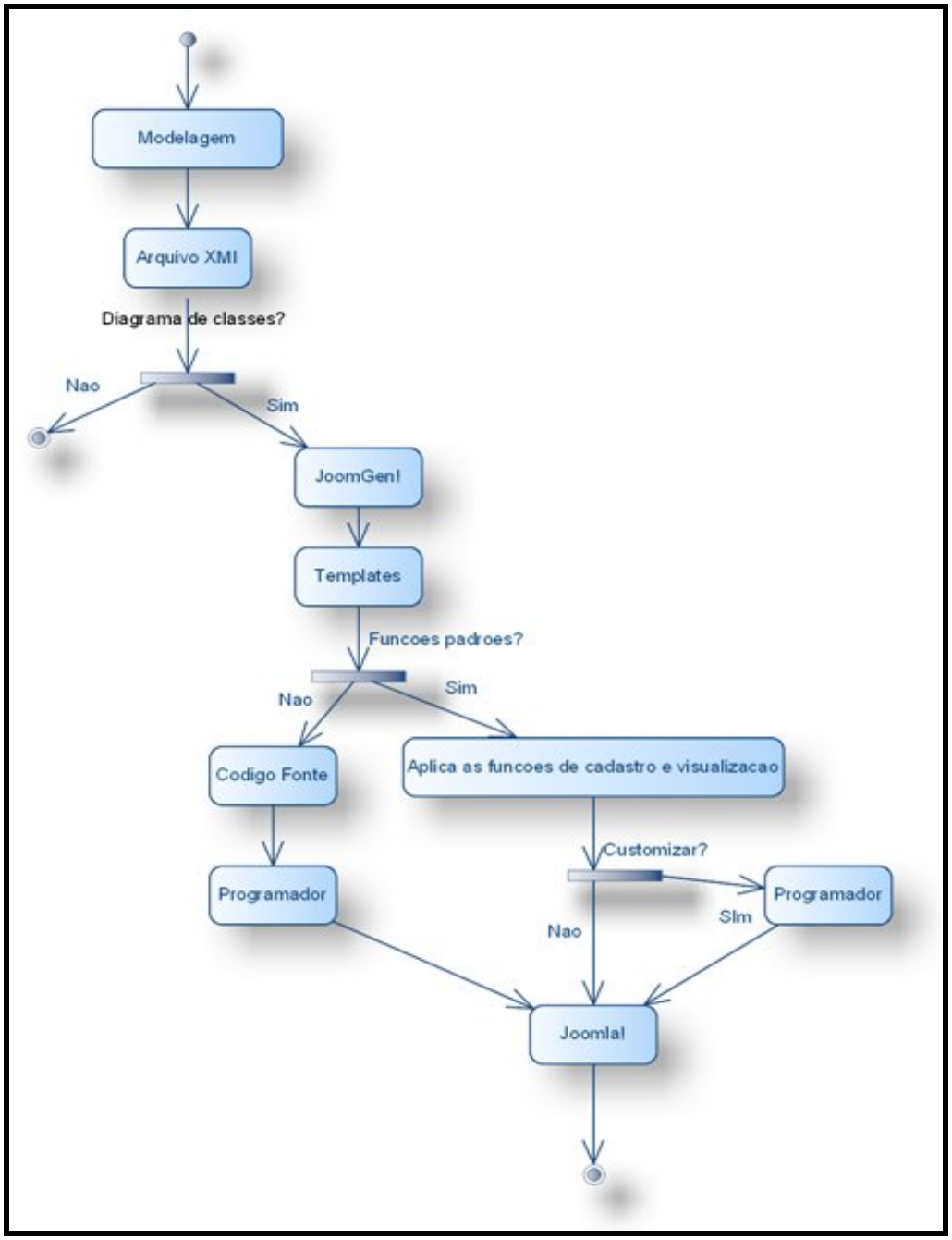

Figura 12: Diagrama de atividades

4.3 LINGUAGEM DE PROGRAMAÇÃO

A linguagem de programação escolhida para o desenvolvimento da ferramenta foi o PHP. Esta é uma linguagem estruturada para a criação de *scripts* e é executada em um servidor Web tendo por finalidade a execução de programas dinâmicos dentro de uma página HTML (WELLING; THOMSON, 2003).

O PHP é uma linguagem *Open Source*, ou seja, qualquer desenvolvedor pode ter acesso ao código-fonte do produto, alterar e redistribuir o mesmo.

A versão do PHP definida para o desenvolvimento é o PHP 5, pois conta com as seguintes funcionalidades fundamentais para o desenvolvimento do JoomGen:

a) o *Zend Engine* 2: provê suporte a métodos privados e protegidos, classes abstratas e interfaces;

b) suporte a reutilização e manipulação de arquivos xml por meio da biblioteca *libxml2* e *simple_xml*;

c) exceções e tratamento de exceções (CONVERSE; PARK; MORGAN, 2004).

4.4 PROGRAMAÇÃO DE COMPONENTES PARA O FRAMEWORK JOOMLA

Os componentes são programas executáveis, que são instalados dentro do CMS Joomla. O desenvolvimento desses programas requer do programador alguns pré-requisitos:

a) conhecimentos em programação PHP e orientação a objetos;

b) conhecimento em bancos de dados MySQL;

c) noções de XML.

Nos tópicos a seguir, serão abordadas, algumas particularidades da programação para o ambiente Joomla em relação à programação WEB tradicional, essencial para o entendimento do código de saída, que é gerado pela ferramenta Joomgen.

4.4.1 O Padrão do Joomla

Uma equipe de desenvolvimento, principalmente as que trabalham com *Open Source*, ao desenvolver um software de maneira que assegure estabilidade, segurança e facilidades em sua manutenção, precisa adotar certas convenções quanto à sua arquitetura de desenvolvimento.

Para criar um componente de cadastro de produtos por meio do Joomla, por exemplo, utiliza-se no mínimo seis arquivos, cada qual com sua finalidade específica. Considerando que o nome do componente seja “produtos”, seus arquivos são:

a) admin.produtos.php: este arquivo define toda a lógica de negócios dos componentes e tem a finalidade de controlar todas as ações do mesmo. Ele será chamado quando o componente for acessado na parte administrativa (*Back End*) do Joomla.

b) admin.produtos.html.php: arquivo que separa a apresentação (as telas) do código do componente que será acessado pelo *Back End* do Joomla.

c) produtos.class.php: este arquivo conterá as classes abstratas. É ele que ligará o componente com a camada de persistência (banco de dados).

d) toolbar.produtos.php: arquivo que tem por finalidade controlar o menu do componente.

e) toolbar.produtos.html.php: este arquivo conterá as chamadas dos menus para cada parte do componente. O mesmo será invocado pelo arquivo toolbar.produtos.php

f) produtos.xml: o conteúdo deste arquivo conterá os parâmetros definidos para instalação do programa, como as pastas padrões, os arquivos contidos no componente e a instalação do banco de dados por meio de *queries*[25].

A união desses seis arquivos reunidos e compactados (formato *.zip) constituirão o componente que estará pronto para ser instalado (JOOMLA BRASIL, 2007).

4.4.2 Acesso e Gravação dos Dados

O Joomla possui um banco de dados comum para todos seus componentes, no entanto, ele precisa ser acessado uma vez a cada requisição de páginas. Para isto, foi implementado o acesso aos dados por meio de objetos (camada de persistência). Este acesso fornece um conjunto de funções que possibilitam a manipulação dos dados, como a requisição de consultas e obtenção dos resultados por meio de classes de Orientação a Objetos. Estas funções são independentes e flexíveis de tal forma possibilita que o usuário (programador) instale diversas copias do CMS na mesma estrutura (LE BLANC, 2007).

Além disso, o Joomla tem um padrão próprio para as classes que manipulam as tabelas, nas quais podem ser gravados dados visualizados, atualizados e excluídos usando as funções pré-existentes sem que o programador precise desenvolvê-las.

[25] Consulta/Pesquisa em banco de dados.

Também podem ser adicionadas lógicas, de maneira com que, se um registro é apagado em uma tabela “mãe”, o mesmo será excluído em cascata em suas tabelas “herdeiras” (filhas).

Isso facilita tanto na manipulação de dados e reaproveitamento de código, quanto na segurança por meio da camada de persistência dos dados, permitindo assim uma maior integridade e confiabilidade resultante de muitos casos de sucesso, comumente encontrados em aplicações Java[26].

4.4.3 Segurança e acesso a variáveis

Por ser uma aplicação w*eb* de acesso público, o Joomla necessita de implementações técnicas para a sua proteção contra as vulnerabilidades e ataques provindos da rede. Pensando nisso, a equipe de desenvolvimento do Joomla criou métodos para que os scripts e variáveis dos componentes somente possam ser invocados dentro do *framework* do Joomla e pelos seus usuários autorizados.

Esses *scripts* são comumente usados por *crackers*[27] para a execução ilegal de informação, captura de dados, negação de serviços entre outras atrocidades em sítios que utilizam tecnologias Web. O Joomla, por sua vez, conta com uma vasta equipe de desenvolvedores especializados em segurança, para prever e minimizar esses ataques.,

4.4.4 Joomla Framework

Dentro do Joomla, existem inúmeras funções e classes pré-programadas. Por meio delas é possível controlar todo o fluxo do negócio, como botões, formulários,

[26] Podem ser encontrados em http://java.sun.com
[27] Cracker é o termo usado para designar quem quebra um sistema de segurança, de forma ilegal ou sem ética.

tabelas e variáveis. Por exemplo, é possível carregar uma lista simplesmente invocando o evento: *Objeto->loadObjetctList* sem precisar que o programador escreva as linhas de código necessárias para que este seja feito integrando PHP, HTML e MySQL.

Além deste evento, pode-se encontrar inúmeros outros, basta consultar sua API[28] on-line, onde são disponibilizadas todas as classes, métodos, variáveis, construtores, funções, pacotes do núcleo do Joomla com finalidades diversas como: manipulações de artigos, armazenamento de dados em *cache*, funções de compatibilidade, funções de manipulações a bancos de dados, funções de manipulações de documentos, eventos HTML, utilitários entre outros.

4.4.5 Definição de parâmetros por meio de XML

Em vez de criar uma tabela separada para os parâmetros de cada extensão, o Joomla separou em arquivos XML para facilitar a parametrização e manipulação das particularidades de cada aplicação.

O Joomla, ao detectar o arquivo XML do componente que será instalado imediatamente, interpreta seus dados como caminho para instalação, nome do componente, nome do autor, os arquivos que serão apontados para seus componentes, as imagens, entre outros.

A Figura 13 mostra como é organizado um arquivo XML de instalação de um componente Joomla:

[28] Disponível em: http://api.joomla.org

```
<?xml version="1.0" ?>
<mosinstall type="component">
<name>moda</name>
<creationDate>03/01/2007</creationDate>
<author>Rodrigo Smania Spillere</author>
<copyright>This component in released under the GNU/GPL License</copyright>
<authorEmail>rodrigospillere@gmail.com</authorEmail>
<authorUrl>www.tempusti.com</authorUrl>
<version>1.0</version>
<files>
<filename>moda.php</filename>
<filename>moda.class.php</filename>
<filename>imagem.php</filename>
</files>
<install>
<queries>
    <query>
    DROP TABLE IF EXISTS `#__moda`;
    </query>
    <query>
    CREATE TABLE `#__moda` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `descricao` TEXT NOT NULL,
    `referencia` TEXT NOT NULL,
    `tamanhos` TEXT NOT NULL,
    `cores` TEXT NOT NULL,
    `categorias` TEXT NOT NULL,
    `published` TINYINT(1) NOT NULL,
    `id_categoria` INT NOT NULL,
    `id_cor` INT NOT NULL,
    `id_tamanho` INT NOT NULL,
    `image_path` TEXT NOT NULL,
    `preco_varejo` TEXT NOT NULL,
    `preco_atacado` TEXT NOT NULL,
    PRIMARY KEY (`id`)
    )
```

Figura 13: Arquivo de instalação XML
Fonte: JOOMLA (2008)

No próximo tópico serão abordadas as técnicas utilizadas pela ferramenta JoomGen para a extração de informações nos arquivos do padrão XMI.

4.5 FERRAMENTA CASE UML

A ferramenta case escolhida para o desenvolvimento do trabalho foi a Star UML por três motivos principais:

a) ser uma ferramenta *freeware*, ou seja, licença de utilização gratuita;

b) suportar exportações dos diagramas em arquivo XMI 1.1;

c) facilidade na criação de diagramas.

Porém, pode ser utilizada outra ferramenta, desde que contemple o suporte a exportação de arquivos no formato XMI 1.1.

4.6 PARTICULARIDADES NA DIAGRAMAÇÃO UML

Foram definidas algumas particularidades em relação à criação de diagramas de classes neste trabalho, com a finalidade de aumentar a funcionalidade da ferramenta JoomGen e adentrar aos padrões especificados pela Joomla Foundation, entre elas:

a) iniciar o nome das classes em maiúsculo: simplesmente por padrão, caso o programador inicie o nome da classe em minúsculo, não ocorrerá interferências no funcionamento do componente gerado pelo JoomGen;

b) configurar o tipo dos atributos: esta é uma parte que deve-se tomar muito cuidado, pois, por meio do tipo de dado especificado na ferramenta, será criada a base de dados, logo, esses devem ser suportados pelo banco de dados MySQL. Caso algum atributo não tenha o seu tipo de dado configurado ou tenha configurado um tipo de dado que não existe no mySQL, ao instalar o componente gerado pelo JoomGen no Joomla, o mesmo apresentará erro de banco de dados;

c) *atibutos id* e *published*: estes devem estar contidos na classe principal do componente e ambos com o tipo de dado int;

d) chaves primárias: para atribuir uma chave primária deve-se configurar a função *Changeability* como ADDONLY no atributo que receberá a chave primária conforme mostra a Figura 14.

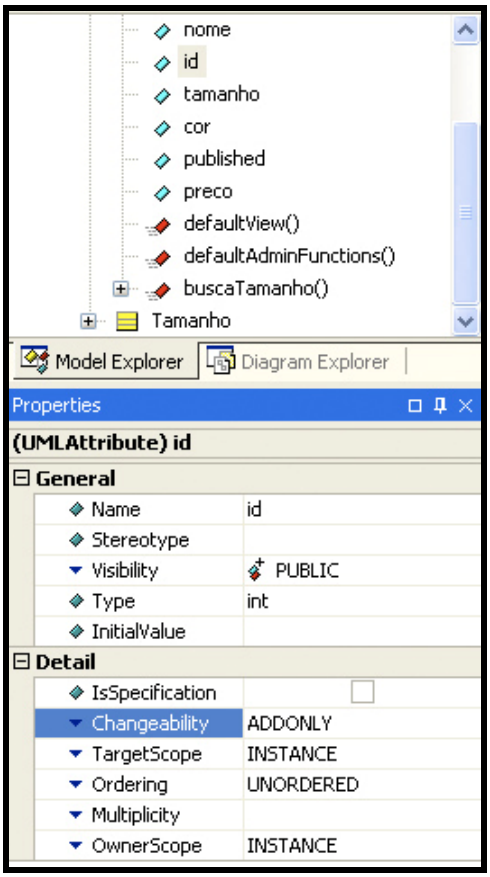

Figura 14: Configuração de chave primaria
Fonte: Star UML (2008)

e) deve-se definir o estereótipo *executable* para a classe principal (aquele que será executável no componente), conforme a Figura 15.

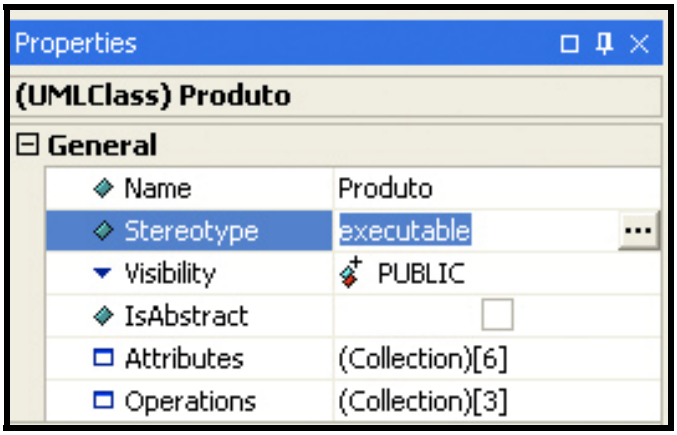

Figura 15: Classe principal
Fonte: Star UML (2008)

A Figura 16 mostra um exemplo de diagrama de classes construído de acordo com as particularidades da ferramenta JoomGen.

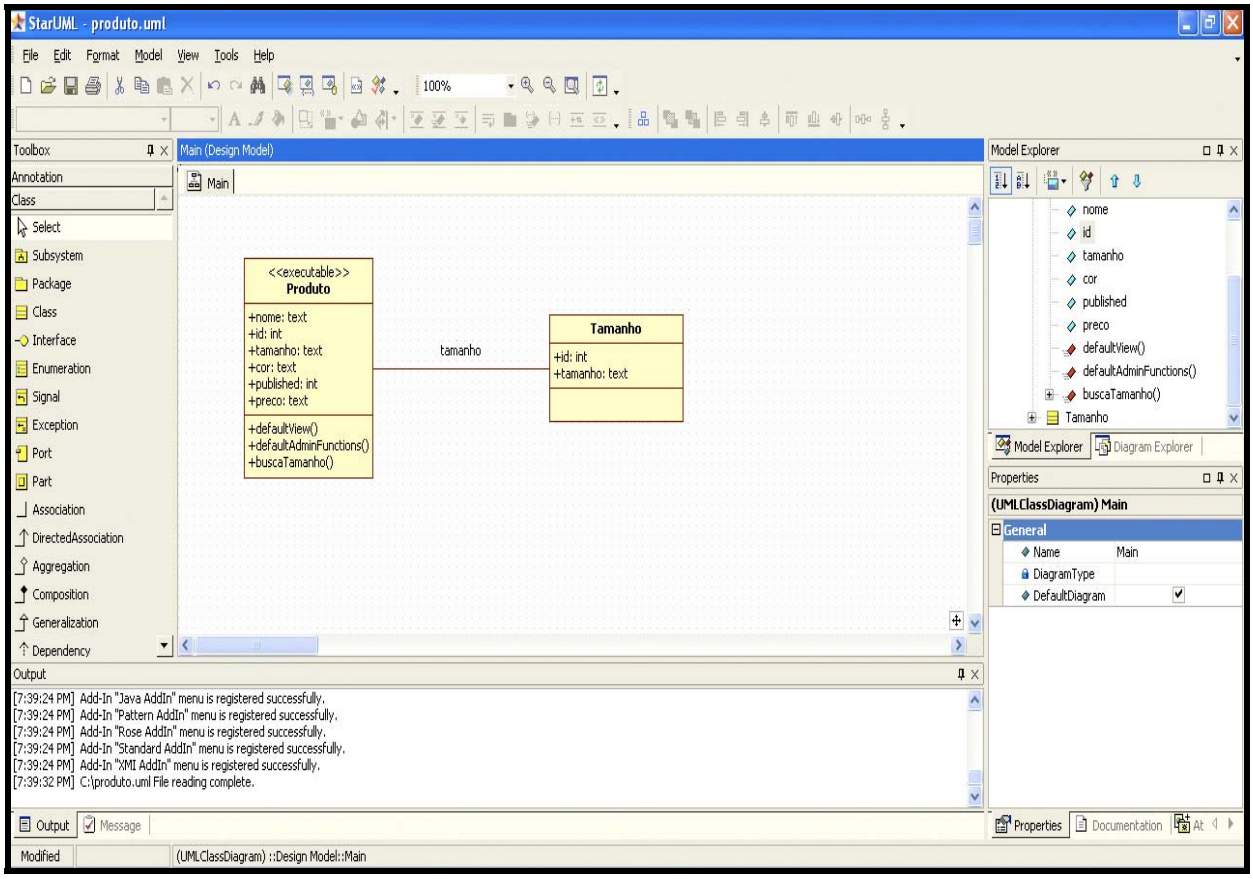

Figura 16: Diagrama de Classes
Fonte: Start UML (2008)

O último passo para a construção do diagrama é salvar as alterações para que possa ser exportado o arquivo que será aprofundado no próximo tópico.

4.7 EXTRAÇÃO DE INFORMAÇÕES DOS DIAGRAMAS DE CLASSES UML

Após criar o diagrama na ferramenta Star UML, deve-se selecionar a opção *File – Export – XMI ...*, conforme a Figura 17. Após isso, o Star UML exporta o diagrama no formato XMI 1.1.

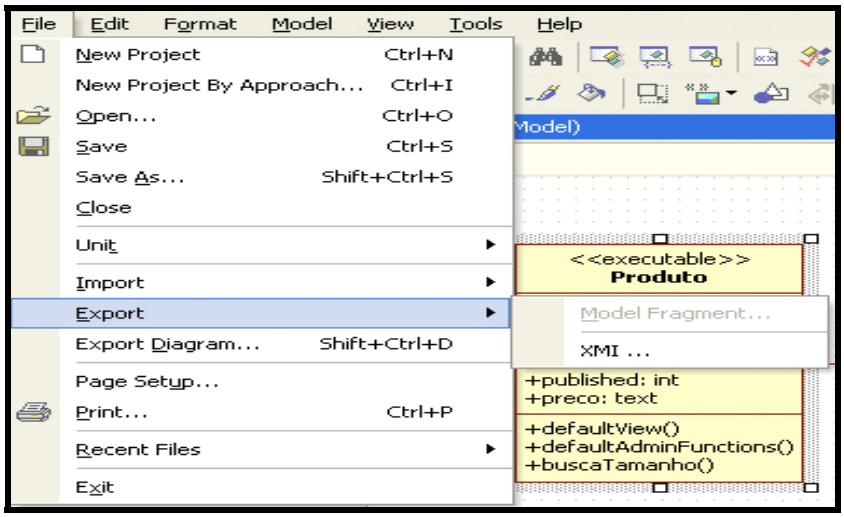

Figura 17: Exportação arquivo XMI
Fonte: Start UML (2008)

Ao analisar o arquivo gerado pelo diagrama, podemos perceber, que ele gera um XML com algumas particularidades como mostra a Figura 18.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<XMI xmi.version = "1.1" xmlns:UML="href://org.omg/UML/1.3" timestamp = "Fri May 02 5:57:33 2008">
<XMI.header>
  <XMI.documentation>
    <XMI.owner></XMI.owner>
    <XMI.contact></XMI.contact>
    <XMI.exporter>StarUML.XMI-Addin</XMI.exporter>
    <XMI.exporterVersion>1.0</XMI.exporterVersion>
    <XMI.notice></XMI.notice>
  </XMI.documentation>
  <XMI.metamodel xmi.name = "UML" xmi.version = "1.3"/>
 </XMI.header>
<XMI.content>
<UML:Model xmi.id="UMLProject.1">
  <UML:Namespace.ownedElement>
    <UML:Model xmi.id="UMLModel.2" name="Design Model" visibility="public" isSpecification="false" namespace="UMLProject.1" isRoot="false" isLeaf="false"
isAbstract="false">
      <UML:Namespace.ownedElement>
        <UML:Class xmi.id="UMLClass.3" name="produto" visibility="public" isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false"
isAbstract="false" participant="UMLAssociationEnd.18" isActive="false">
          <UML:Classifier.feature>
```

Figura 18: Arquivo XMI

Não compete ao JoomGen analisar o arquivo com a finalidade de verificar se o mesmo está de acordo com os *Schemas* e DTDs impostos pelo padrão XMI, pois

isto é o papel da ferramenta case UML. O papel do JoomGen é interpretar este arquivo e transformar em dados para que seja feita a geração dos códigos fontes.

Um diagrama de classes é composto por alguns componentes, como vistos no item 3.3.2. O arquivo XMI mapeia estes componentes através de *tags xml*, cada qual com sua funcionalidade, por exemplo:

a) tipo de diagrama: a *tag* responsável pela definição do tipo do diagrama UML é o <UML:Model>. Esta somente é utilizada na ferramenta JoomGen por motivo de consistência. Caso o diagrama especificado não seja o diagrama de classes, o programa importador retornará um erro. A figura 19 apresenta um exemplo da utilização da *tag UML:Model* em um diagrama de classes.

```
<UML:Model xmi.id="UMLProject.1">
  <UML:Namespace.ownedElement>
    <UML:Model xmi.id="UMLModel.2" name="Design Model" visibility="public"
isSpecification="false" namespace="UMLProject.1" isRoot="false" isLeaf="false"
isAbstract="false">
```

Figura 19: Tag UML:Model

b) classes: as informações referentes a cada classe do diagrama, são definidas na *tag <UML:Class>* conforme mostra a figura 20.

```
      <UML:Class xmi.id="UMLClass.3" name="produto" visibility="public"
isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false"
isAbstract="false" participant="UMLAssociationEnd.18" isActive="false">
```

Figura 20: Tag UML:Class

c) atributos: as informações acerca de cada atributo, ficam localizados dentro da tag *<UML:Class>* referente à classe que o mesmo pertence. A *tag* referente aos atributos é definida pela *tag <UML:Attribute>*, dentro de cada atributo está definido a classe que o mesmo pertece, por meio do parâmetro *owner* comomostra a figura 21.

```
<UML:Attribute xmi.id="UMLAttribute.4" name="nome" visibility="public"
isSpecification="false" ownerScope="instance" changeability="changeable"
targetScope="instance" type="X.23" owner="UMLClass.3"/>
```

Figura 21: Tag UML:Attribute

d) métodos e parâmetros: os métodos são definidos pela *tag* <*UML:Operation*> e os parâmetros de cada método são definidos pela *tag* <*UML:Parameter*> como apresenta a figura 22.

```
<UML:Operation xmi.id="UMLOperation.10" name="defaultView" visibility="public"
isSpecification="false" ownerScope="instance" isQuery="false" concurrency="sequential"
isRoot="false" isLeaf="false" isAbstract="false" specification="" owner="UMLClass.3"/>
```

Figura 22: Tag Uml:Attribute

e) associações e conexões: as associações que interligam os diagramas são definidas pelo nome e por sua conexão à classe onde partiu e à classe destino, conforme a figura 23.

```
<UML:Association xmi.id="UMLAssociation.17" name="tamanho" visibility="public"
isSpecification="false" namespace="UMLModel.2">
  <UML:Association.connection>
    <UML:AssociationEnd xmi.id="UMLAssociationEnd.18" name="" visibility="public"
isSpecification="false" isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable" association="UMLAssociation.17"
type="UMLClass.3"/>
```

Figura 23: Tag Uml:Association

f) tipos de dados: os tipos de dados são definidos no XMI com a *tag* <*UML:DataType*> como é mostrada pela figura 24.

```
<UML:DataType xmi.id="X.23" name="text" visibility="public" isSpecification="false"
isRoot="false" isLeaf="false" isAbstract="false"/>
```

Figura 24: Tag Uml:Datatype

g) etereótipos: são definidos pela *tag* <*UML:Stereotype*> mostrando o valor e apontando para o id do diagrama correspondente, conforme mostra a figura 25.

```
<UML:Stereotype xmi.id="X.20" name="designModel" extendedElement="UMLModel.2"/>
<UML:Stereotype xmi.id="X.21" name="executable" extendedElement="UMLClass.3"/>
```

Figura 25: Tag Uml:Stereotype

A partir desses dados, é possível fazer o mapeamento das informações advindas do diagrama de classes e, com a ajuda da classe SimpleXMLElement do PHP5, consegue-se extraí-las do arquivo XML para que possam ser manipuladas pela ferramenta JoomGen, conforme a Figura 26.

```
16  * Construtor - Responsável por carregar o arquivo XMI 1.3
17  */
18  function __construct($xmi) {
19  $arquivo = realpath($xmi);
20          if (!file_exists($arquivo)) {
21              echo "<br>Arquivo não Encontrado!!</br>";
22          }
23          else {
24              //carrega o conteúdo do arquivo para uma string
25              $xmlData = file_get_contents($arquivo);
26              //gera um objeto do tipo SimpleXMLElement
27              $this->xml = new SimpleXmlElement($xmlData);
28              if  (!$this->xml->xpath('//UML:Class')){
29                  echo "<br> este diagrama não é suportado pelo Joomgen! ";
30                  exit;
31              }
32              foreach ($this->xml->xpath('//UML:Class') as $clas) {
33                 $className = $class['name'];
34              }
35          }
36  }
```

Figura 26: Construtor

Foi implementado na ferramenta JoomGen, um método para mostrar essas informações na tela de forma que o programador verifique se a ferramenta tratou corretamente os dados depois de importados. O método é mostrado na Figura 27, onde

pode-se perceber algumas linhas de código, na linha 42 é onde ele chama o diagrama importado pelo construtor, na linha 44 ele utiliza do método xpath da classe SimpleXMLElement para percorrer as classes e mostra o resultado na tela com o comando *echo* na linha 46.

```
41 function info(){
42         $xml = $this->xml;
43         // Captura a tag UML:Class do arquivo XMI
44         foreach ($xml->xpath('//UML:Class') as $clas) {
45             echo "<hr />";
46             echo "<br /><b> Classe: </b>".$clas['name'];
47             $className=$clas['name'];
48             $this->classe_nome = $className;
49             $classId=$clas['xmi.id'];
```

Figura 27: Função info()

No próximo item será mostrado como foi estruturada a ferramenta JoomGen.

4.8 A ARQUITETURA DA FERRAMENTA JOOMGEN

O JoomGen tem um padrão próprio para o desenvolvimento. De acordo com muitos estudos de padrões de projetos, o seu modelo se assemelha ao MVC, porém, por não utilizar a camada de persistência de dados, difere-se um pouco do padrão.

A estrutura física do JoomGen é dividida em vários diretórios.

a) class: as classes do programa;

b) configs: arquivos de configurações;

c) functions: algumas funções úteis para o programa, como função de compactar e gerar arquivo de configuração;

d) input: local onde serão armazenados os diagramas uploadados;

e) interfaces: telas do programa;

f) output: local onde fica o código gerado pela ferramenta, o produto final, que será compactado e disponibilizado para o usuário fazer download;

g) templates: arquivos de modelos para a geração dos códigos;

h) arquivo core.php: núcleo do programa, faz a interligação dos arquivos, classes, templates, funções, telas, etc..

A Figura 28 mostra a estrutura de diretórios do JoomGen.

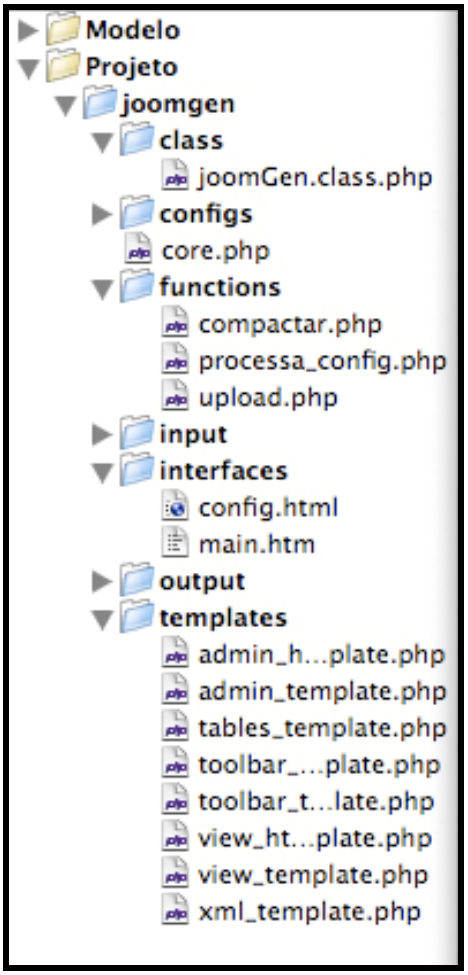

Figura 28: Pastas e arquivos do JoomGen

Uma das grandes vantagens de se ter uma arquitetura como visto na Figura 28, é a flexibilidade, pois, por meio dela, pode-se trabalhar separadamente a parte visual do programa, a parte de modelagem com a parte de programação. Com a separação dos

templates, é possível adicionar outras funcionalidades, como por exemplo, a geração de códigos para outras plataformas, ou então adicionar funções de internacionalização dos códigos.

4.9 GERAÇÃO DE COMPONENTES POR MEIO DE DIAGRAMAS UML

Foi mostrado no item 4.8 como o JoomGen faz para extrair as informações dos diagramas UML e transportá-las para dados. Neste item será abordado como a ferramenta faz para trabalhar essas informações a fim de criar os scripts de bancos de dados, códigos de persistência, lógicas de negócios, interfaces, entre outros.

O JoomGen, por padrão, tem seu mecanismo de leitura separado dos mecanismos de *templates*, como foi mostrado no item 4.8. Também foi visto no item 4.5.1, que um componente Joomla de apenas uma classe é composto de no mínimo 6 arquivos e, a medida que aumenta o número de classes, o número de arquivos também aumenta. Em seu padrão, o Joomgen conta com 7 *templates* diferentes, sendo que o mesmo suporta criação de outros, desde que os mesmos sigam o seu padrão de desenvolvimento.

A Figura 29 mostra o método da classe Generate do Joomgen, responsável por chamar o *template* e gerar o arquivo. Este método captura as informações do arquivo de configuração e, logo após, passa para o *template*, esse retorna uma *string* para o método e logo após é gravado no arquivo como mostra a linha 113 da figura 29.

```
function makeToolbar() {
        require("configs/var_config.php");
        require_once("templates/toolbar_template.php");
        $arquivo = "$path/admin/toolbar.$component_name_lower.php";
        file_put_contents($arquivo, $toolbar_str);
        if($arquivo){
          echo "<br /> toolbar.$component_name_lower.php criado com sucesso!";
        }else{
          echo "<br /> Erro: Impossível criar toolbar.$component_name_lower.php";
        }
}
```

Figura 29: Função makeToolbar

A Figura 30 mostra o arquivo de *template* toolbar.php que a função makeToolbar utilizou.

```
<?
require("configs/var_config.php");
$toolbar_html_str = <<<PHP
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
class TOOLBAR_$component_name_lower {
      function _NEW() {
              JToolBarHelper::save();
              JToolBarHelper::apply();
              JToolBarHelper::cancel();
      }
      function _DEFAULT() {
              JToolBarHelper::title( JText::_( '$component_name' ), 'generic.png' );
              JToolBarHelper::publishList();
              JToolBarHelper::unpublishList();
              JToolBarHelper::editList();
              JToolBarHelper::deleteList();
              JToolBarHelper::addNew();
      }
}
?>
PHP;
?>
```

Figura 30: Template do arquivo toolbar

Na classe Generate existem 7 métodos para geração de arquivos baseadas em *templates*, estas são:

a) makeXml: responsável por criar os arquivos de instalação e o script do banco de dados;

b) makeToolbar: criação de barras de ferramentas[29];

c) makeToolbarHtml: interfaces do arquivo de barras de ferramentas;

d) makeTables: método responsável pela persistência dos dados;

e) makeAdmin: método de lógica de negócio, que irá controlar a área administrativa;

f) makeAdminHtml: interfaces do arquivo anterior;

g) makeView: método responsável pela lógica do frontEnd do componente;

h) makeViewHtml: interfaces do arquivo anterior.

No próximo tópico será mostrado os procedimentos para a geração de componentes com o JoomGen.

4.10 GERAÇÃO DE COMPONENTES POR MEIO DA FERRAMETA JOOMGEN

Para fazer a criação do código é necessário um computador, com sistema operacional, de preferência software livre, como Linux, freeBSD, entre outros. Nos testes foi usado o sistema Mac OSX 10.4.

Primeiramente deve ser instalado o servidor Apache + MySQL + PHP. No caso do MAC, foi instalada a ferramenta MAMP que contempla todos esses servidores, sem que seja preciso efetuar a configuração de cada um. Na mídia que acompanha este trabalho, se encontram a instalação de 3 ambientes Apache + MySQL + PHP, um para os sistemas Windows, um para os sistemas Linux e outro para os sistemas Mac OSX. A instalação no ambiente MAC OSX e Windows será mostrado no arquivo Licence.txt na mídia que acompanha este trabalho.

[29] vide figura 29

Após efetuar a instalação, devemos copiar a pasta Joomgen para a pasta do apache, e iniciar os serviços do Apache, MySQL e PHP, conforme a figura 31.

Figura 31: Iniciando o AMP
Fonte: MAMP (2008)

Iniciados os serviços na máquina local, basta abrir o navegador de internet e entrar com o seguinte endereço http://localhost/joomgen/. Em seguida, abrirá a tela inicial do JoomGen.

Como exemplo será gerado um componente de catálogo de produtos de uma empresa especializada em roupas para animais chamado de Animais.

Primeiramente deve-se criar o diagrama de classes referente ao programa. Em seguida, serão adicionadas três classes ao Star UML e populados com os atributos e métodos, conforme mostrado na figura 32.

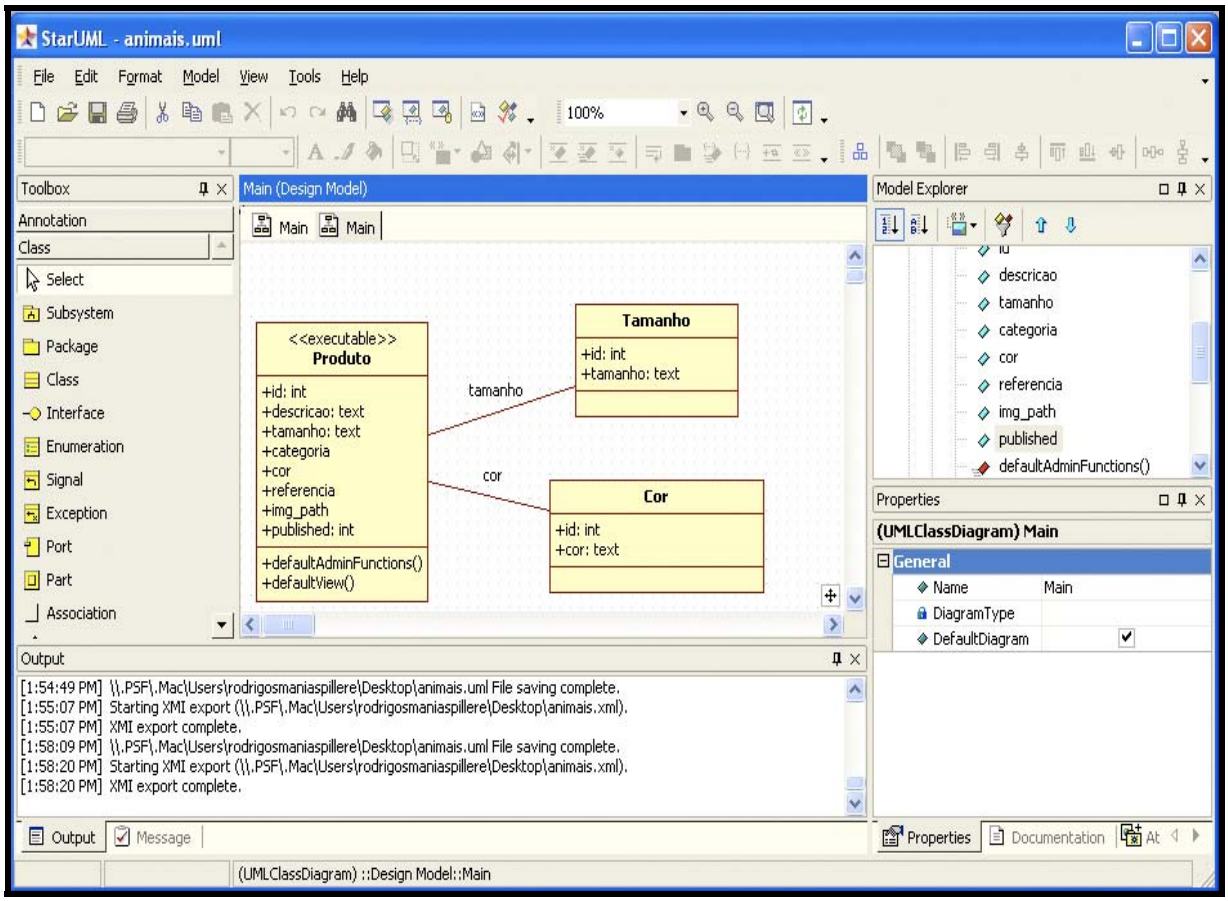

Figura 32: Diagrama de classes do componente Animais
Fonte: Star UML (2008)

Feito isso, no navegador basta entrar no endereço onde foi instalado o Joomgen. Ao entrar no endereço especificado será exibido o formulário mostrado pela figura 33, onde deve-se preencher os seguintes campos.

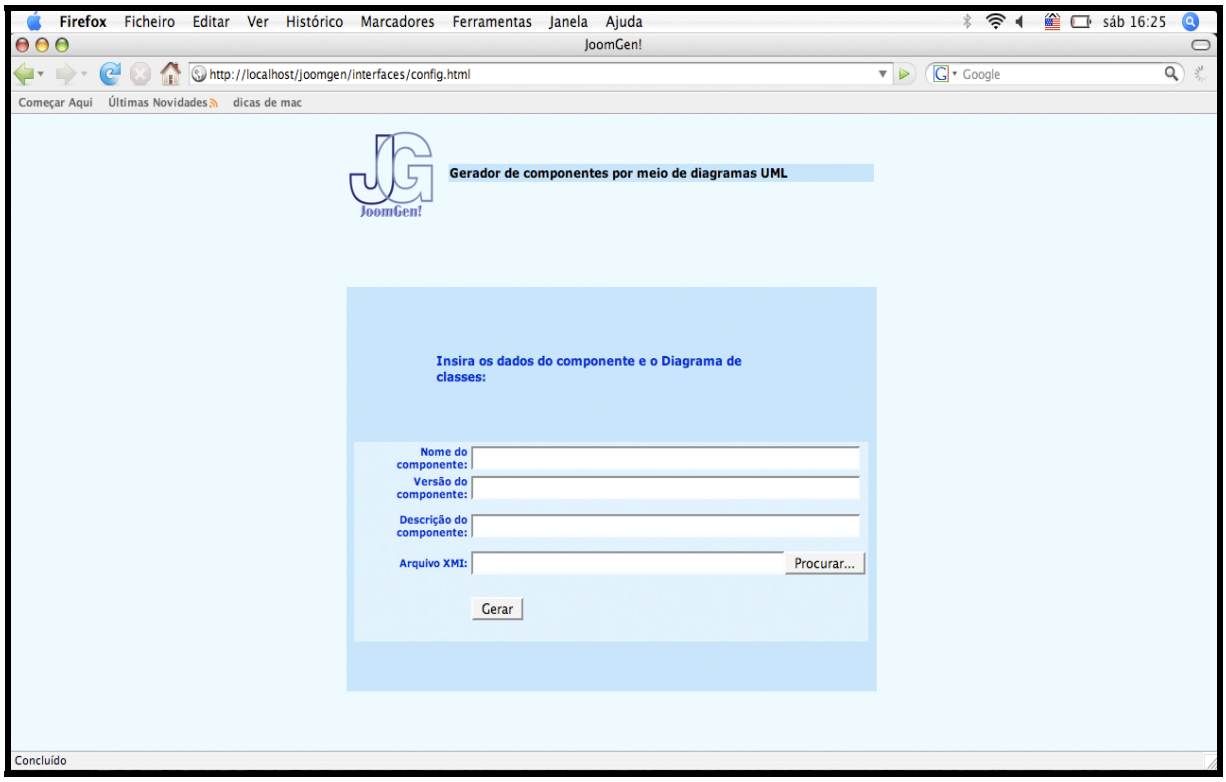

Figura 33: Tela inicial do JoomGen

a) nome do componente: este deve ser preenchido com o nome que será dado ao componente que será gerado. No experimento será preenchido com o valor “Animais”;

b) versão do componente: deve ser preenchido com a versão do componente gerado. No experimento será preenchido o valor “v0.1”;

c) descrição do componente: uma descrição curta do componente. No experimento será preenchido com “Componente de Catalogo de roupas para animais”;

d) arquivo XMI: buscar o local no disco onde foi exportado o arquivo de diagramas UML.

Após o preenchimento dos dados, basta clicar no botão Gerar, o mesmo irá processar e retornar a mensagem mostrada pela Figura 34.

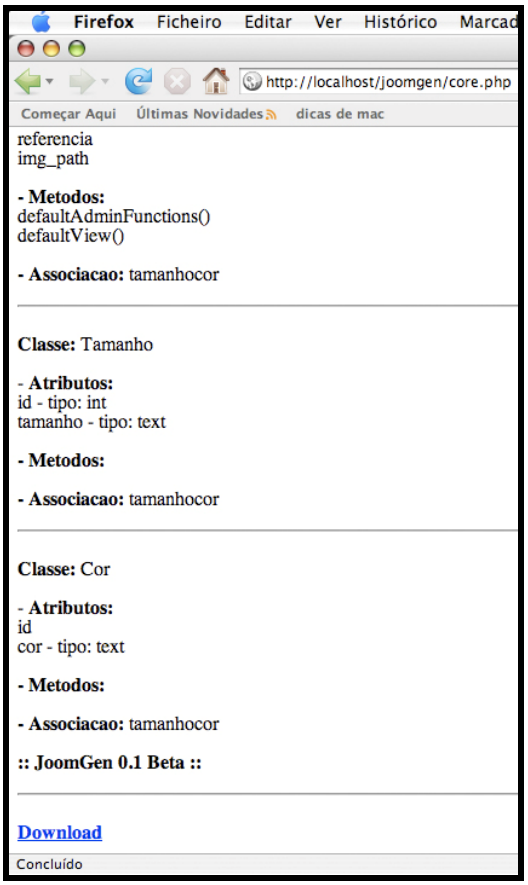

Figura 34: Geração do componente.

A última parte é baixar o componente gerado, este disponibilizado em formato .zip, bastando clicar em download e selecionar o lugar a ser salvo no computador local.

Ao descompactar o arquivo, percebe-se que foi gerado uma pasta chamada com_animais constituída pelos seguintes arquivos: animais.xml, animais.php, animais.html.php, admin.animais.php, admin.animais.html.php, toolbar.animais.php, toolbar.animais.html.php, produto.php, tamanho.php, cor.php, conforme mostrado na Figura 35.

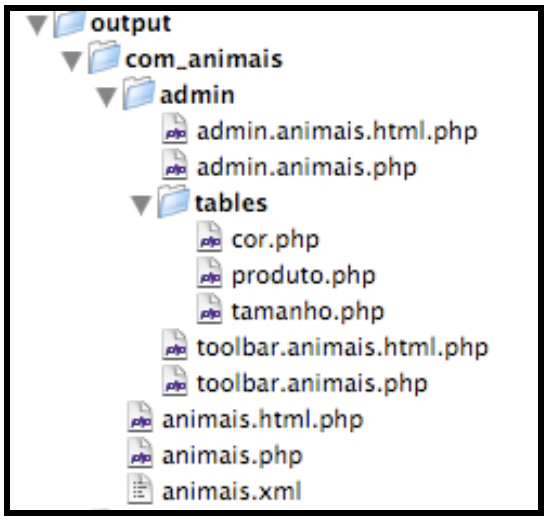

Figura 35: Componente gerado

Após, o programador apenas precisará editar o arquivo admin.animais.php para programar as funções da área administrativa e o arquivo admin.joomgen.html.php para programar o formulário, toda a parte de banco de dados, classes e menus foram criados pelo JoomGen. A versão final do componente animais pode ser visualizado pela Figura 36.

Figura 36: Visualização do componente Animais

4.11 GERAÇÃO DE COMPONENTES COMPLETOS COM O JOOMGEN

Para facilitar o trabalho dos programadores, o Joomgen conta com uma série de funções pré-programadas que são muito usadas pelos componentes *web*. Estas são as funções de novo cadastro, exclusão, edição, salvar, aplicar e busca.

Para utilizar essas funções, basta adicionar na classe marcada com o estereótipo <<executable>> os métodos *defaultAdminFunctions* e *dafaultView*, conforme a Figura 37.

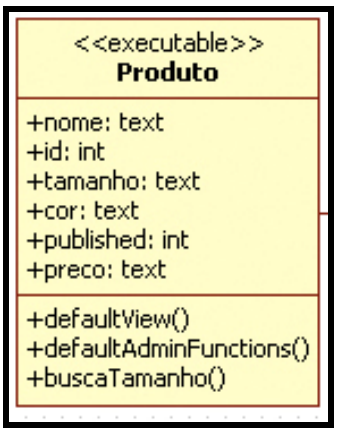

Figura 37: Funções padrão

O diagrama da figura 37 é referente a um cadastro de produtos para uma loja de departamentos. Ao finalizar a diagramação, deve ser importado pelo Joomgen que se encarregará da programação das principais funções do mesmo, ou seja, após o componente ser gerado pelo Joomgen, ele estará pronto para ser importado para o CMS Joomla.

4.12 APLICANDO O COMPONENTE GERADO NO JOOMLA

Para finalizar a geração do componente no Joomla, deve-se logar no painel de administração e acessar o menu *extensions* e, em seguida em *Install/Uninstall*. Com isso, abrirá a tela representada na Figura 38.

Figura 38: Instalação componentes
Fonte: Joomla (2008)

No campo *Upload Package File*, deverá ser incluído o componente a ser instalado, que deve estar no formato .zip. Logo em seguida, deve-se clicar em *Upload File & Install.*

Feito isso, será exibida uma mensagem de sucesso conforme mostrado na figura 39, ou então uma mensagem de erro, caso o componente gerado tenha algum erro de programação.

Figura 39: Componente instalado com sucesso
Fonte: Joomla (2008)

4.13 RESULTADOS OBTIDOS

Concluída a geração e a importação do componente gerado pelo Joomgem, o componente está pronto para ser utilizado no site ou portal.

O componente de cadastro de produtos gerado no item 4.11 e mostrado nas Figuras 39, 40 e 41, foi gerado automaticamente pelo Joomgem, ou seja, não teve intervenção humana na programação do seu código.

A Figura 40 mostra o cadastro de produtos com a tabela dos produtos cadastrados, os botões de publicação, inserção, exclusão e edição.

Figura 40: Componente produtos – Administração
Fonte: Joomla (2008)

Ao clicar no botão *New* será exibida uma tela de inserção do produto como mostra a Figura 41.

Figura 41: Cadastro de produtos
Fonte: Joomla (2008)

Com o produto cadastrado, todos os produtos marcados como *published* poderão ser visualizados pelos visitantes do site ou portal. O componente irá gerar uma lista com os nomes dos produtos cadastrados e, ao clicar em um produto, o programa irá exibir a descrição do mesmo, como mostrado na Figura 42.

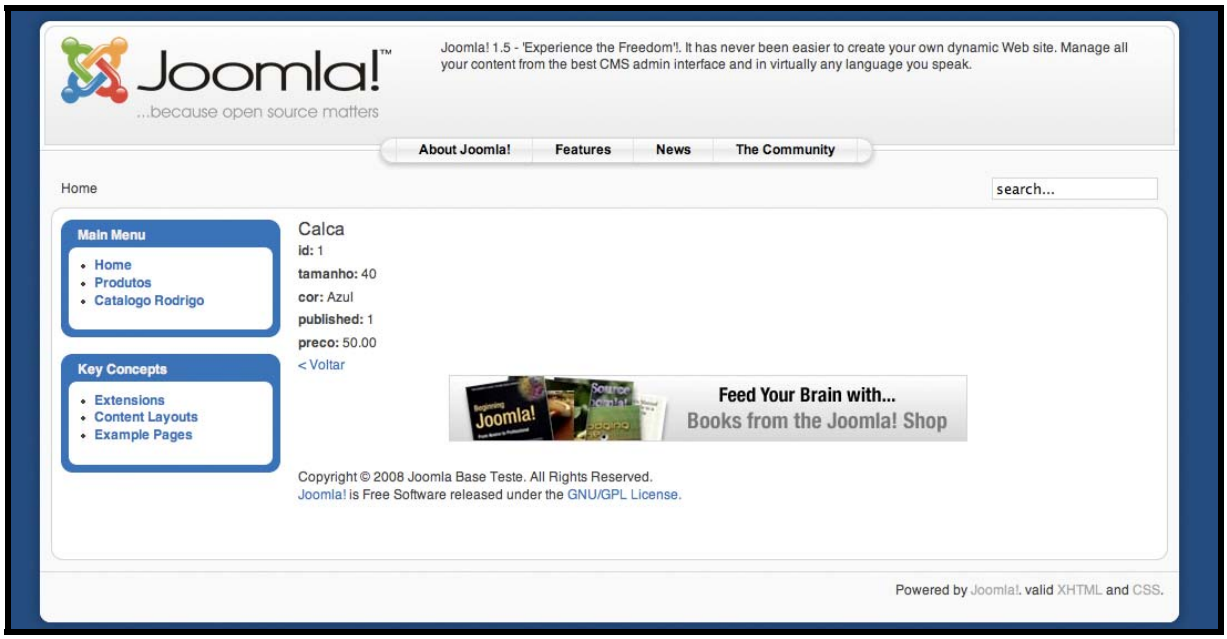

Figura 42: Produto front-end
Fonte: Joomla (2008)

Conforme visto nos dois diferentes códigos gerados, percebe-se que se adquiriu um ganho muito grande com a programação dos componentes para Joomla. No primeiro exemplo, o programador foi abstraído de toda a parte burocrática do código tais com o padrão da API, a parametrização e a criação da base dados. No segundo exemplo, o ganho foi tão grande que não foi preciso editar nenhum código. Após concluir a etapa da diagramação, foi possível gerar o programa completo, por meio das funções pré programadas, descritas no item 4.11. Assim, obteu-se uma excelente reutilização de código.

CONCLUSÃO

Com o desenvolvimento do trabalho, desde o levantamento bibliográfico, definição do tema, desenvolvimento da ferramenta e testes, foram aplicados os conhecimentos adquiridos ao longo da graduação, bem como novas áreas de conhecimento. O estudo teórico do trabalho foi voltado à engenharia e padrões de desenvolvimento de software, cujo é a base para o desenvolvimento do trabalho e a ele foi dedicado boa parte da fundamentação.

O presente trabalho preocupou-se em construir uma maneira em que permitisse abstrair a complexidade dos códigos fontes de um projeto padronizado sem que o conceito do mesmo seja perdido. Para solucionar tal problema, foi desenvolvida uma ferramenta que estimula o desenvolvedor a projetar seu aplicativo e aplicá-lo seguindo os padrões universais utilizando a linguagem UML, que é requisitada por ela. A partir do projeto, a ferramenta abstrai do programador grande parte do código que se destina a padronização do CMS Joomla, fazendo assim, com que todo projeto criado pela ferramenta JoomGen esteja dentro dos padrões definidos para o framework Joomla.

A partir do estudo e compreensão da arquitetura orientada por modelos, foi viável a criação de uma ferramenta capaz de manipular os dados exportados pelos diagramas de classes provenientes de determinadas ferramentas case UML e, a partir desses, fazer a geração automática da estrutura de códigos fonte que envolve um componente para o Framework Joomla.

A arquitetura da ferramenta foi desenvolvida com o intuito de ser flexível o suficiente de maneira que possibilite criar *templates* para outros componentes, definir funções padronizadas e possibilitar a internacionalização de suas interfaces. Com a finalidade de testar as funcionalidades propostas pela ferramenta, foi utilizado um

ambiente com um servidor MAC OSX 10.4. O servidor conta com os serviços Apache 2 + PHP 5 + MySQL 8 + Joomla 1.5 o que proporciona um bom campo para testes com a ferramenta, possui todos os recursos que a mesma demanda.

Os testes com a ferramenta foram executados por um desenvolvedor Joomla. Foi constatado que houve um grande ganho de tempo de desenvolvimento, não só pela parte da padronização dos códigos, mas também facilitando a reutilização de rotinas. Por exemplo, um aplicativo *web* bastante utilizado pelo desenvolvedor em sites e portais coorporativos são os componentes que envolvem o cadastro e disponibilização de informações, como o caso dos catálogos de produtos. Ao criar esses catálogos para diferentes empresas era preciso fazer o projeto, construir o banco de dados e implementar a estrutura do mesmo para cada projeto. Por meio da ferramenta JoomGen, foi possível criar tanto as estruturas do componente quanto a base de dados automaticamente. Para isso, bastou adicionar as funções padronizadas nos diagramas UML.

Assim sendo, observa-se que foram atingidos os objetivos propostos no trabalho, pois os ganhos são visíveis, tais quais, redução de tempo de programação, abstração da complexidade do código do componente, reutilização de códigos, entre outros.

Como sugestões de trabalhos futuros que podem dar continuidade a este projeto bem como de melhorias na ferramenta desenvolvida, são:

a) suporte a outros diagramas, como o diagrama de atividades, estados, utilização;

b) utilização de padrões de códigos que facilitam a internacionalização dos componentes gerados;

c) ampliação das funções pré-definidas fazendo com que aumente a reutilização de códigos;

d) criação de templates para outros gerenciadores de conteúdos como o Drupal, Xoops, PHP-Nuke, bem como outras versões do Joomla;

e) criação de templates para a criação de módulos para o Joomla;

f) integração da modelagem de diagramas com a ferramenta JoomGen.

REFERÊNCIAS

ABITEBOUL, Serge; BUNEMAN, Peter; SUCIU, Dan. **Gerenciando dados na WEB.** Rio de Janeiro: Ed. Campus, 2000.

ANSELMO, Fernando. **PHP e MySQL para windows.** Florianópolis: Bookstore, 2000.

ATWAL, S. **BuildingWebsites with XOOPS**: a step-by-step tutorial. Birminghan: Packt
Publishing, 2006.

BAX, M. P.; PEREIRA, J. C. L**. Introdução à Gestão de Conteúdos**. In: **WORKSHOP
BRASILEIRO DE INTELIGÊNCIA COMPETITIVA E GESTÃO DO CONHECIMENTO**, 2002, São Paulo. Anais. KM Brasil, 2002. n.3.

BEZERRA, Eduardo. Principios **de Análise e Projetos de Sistemas com UML**. Rio de Janeiro: Editora Campus, 2002.

BLANC, Joseph. le. **Learning Joomla 1.5 Extension Developement**. Olton: Packt Publishing Ltd, 2007.

BOOCH, GRADY; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Rio de Janeiro: Ed. Campus, 2000.

BRASIL. **Ministério da Educação e Cultura**. Disponível em: <http://www.mec.gov.br/>. Acesso em 10 de set. 2007.

_____. **Conselho Nacional de Justiça**. Disponível em: <http://www.cnj.gov.br/>. Acesso em 10 de set. 2007.

CONALLEN, Jim. **Desenvolvimento de aplicações Web com UML**. 2. ed Rio de Janeiro: Campus, 2003.

COSTA, Cristina C; GOMES, Franciele D; CAGNIN, Maria I. **Estudo para adaptação de um processo ágil de desenvolvimento de software baseado em modelos**. In: **I Workshop de desenvolvimento rápido de aplicações**, 2007, Porto de Galinhas. Disponível em: <http://reuse.cos.ufrj.br/wdra2007/images/artigos/30385.pdf/>. Acesso em 02 nov. 2007.

DEITEL, H.M; DEITEL, P. J. **Java:** como programar. 4.ed Porto Alegre: Bookman, 2003.

DESCHAMPS, Alexandro; GRAHL, Everaldo A.**Ferramenta para a geração de código a partir da especialização do diagrama de classes**. In: **XIV Seminário de Computação**. Universidade Regional de Blumenau. SEMINCO, 2005, Blumenau.

Disponível em: <http:// www.inf.furb.br/seminco/2004/artigos/112-vf.pdf/>. Acesso em 02 nov. 2007.

DINIZ, Abílio. **Caminhos e escolhas**: o equilíbrio para uma vida mais feliz. 21. Ed. Rio de Janeiro: Editora Campus, 2004.

DYKES, Lucinda. TITTEL, Ed. **XML for Dummies.** 4. Ed. Indiana: Jhon Wiley & Sons, 2005.

FONSECA, V. da. **Introdução às dificuldades de aprendizagem**. Porto Alegre : Artes Médicas, 1985.

FRASER, R. G. Stephen. **Real World ASP.Net:** Building a content managemment system. Heidelberg: Apress, 2002

GAMITO, Rui; CUNHA, Luis A. da; ABREU, **Modelação de workflows com UML e ferramentas declarativas**. In: I **XML:** Aplicações e Tecnologias Associadas, XATA, 2007, Lisboa. Disponível em: <http://xata.fe.up.pt/2007/papers/6.php/>. Acesso em 02 nov. 2007.

GRAF, Hagen. **Building Websites with Joomla**. Olton: Packt Publishing Ltd, 2006.

_____. **Building Websites with Joomla 1.5 Beta 1**. Olton: Packt Publishing Ltd, 2007.

GROSE, J. Timothy. DONEY, C. Gary. BRODSKY A. Stephen. **Mastering XMI Java Programming with XMI, XML, and UML.** Indiana: Jhon Wiley & Sons, 2002.

GONÇALVES, Rodrigo. **Uma ferramenta para aprendizagem de desenvolvimento de sistemas.** Santa Catarina: UFSC, 2004.

HOLZNER, Steven. **Desvendando XML**. Rio de Janeiro: New Riders, 2001.

LAPA, Eduardo.**Gestão de conteúdo:** como apoio do conhecimento. Rio de Janeiro: Brasport, 2004.

JOOMLA BRASIL. **Joomla Brasil**. Disponível em: <http://www.joomla.com.br/>. Acesso
em: 10 Abr. 2007

JOOMLA TEAM. **Joomla CMS**. Disponível em: <http://www.joomla.org/>. Acesso em: 13 Mai. 2008.

MAIA, Natanael E.N.; BLOIS, Ana P; Werner, Cláudia M. **Odyssey-MDA**: uma ferramenta para transformações de modelos UML. In: **XIX Simpósio Brasileiro de Engenharia de Software**, SBES. Universidade Federal de Uberlândia. UFU, 2005, Uberlândia. Disponível em: <http:// www.sbbd-sbes2005.ufu.br/arquivos/odyssey-mda.pdf/>. Acesso em 02 nov. 2007.

MAMBO FOUNDATION. **Mambo CMS**. Disponível em: <http://mambofoundation.org/>. Acesso em: 13 out. 2007.

MEDEIROS, Ernani. **Desenvolvendo software com** UML **2.0**: definitivo. São Paulo: Makron Books, 2004.

MELLOR, Stephen J. et al. **MDA Destilada:** Princípios de arquitetura Orientada por Modelos. Rio de Janeiro: Ciência Moderna, 2005.

MENDES, Antonio. **Programando com** XML. Rio de Janeiro: Elsevier, 2004.

MORDEAUI, Luciana. **Guia de Estilo Web**: Produção e Edição de Notícias Online. São Paulo: Editora Senac, 2000.

PENDER, Tom. **UML Bible.** Indiana: Jhon Wiley & Sons, 2003.

PEREIRA, Marco A. Et al. **Transformando modelos da MDA com o apoio de componentes de software**. In: **Simpósio Brasileiro de Componentes, arquiteturas e reutilização de Software**, SBCARS. Universidade Estadual de Campinas UNICAMP, 2007, Campinas. Disponível em: <http://www.ic.unicamp.br/sbcars2007/tecnicas/presentations/Pereira_TMM.pdf/>. Acesso em 02 nov. 2007.

PHPNUKE.ORG. **PHP-Nuke CMS**. Disponível em: <http://www.phpnuke.org/>. Acesso em: 13 out. 2007.

PILGRIM, Mark. **Mergulhando no PYTHON:** O guia rápido e prático para dominar o Python. Rio de Janeiro: Alta Books, 2004.

PITTS-MOULTIS, Natanya. **XML:** black book. São Paulo: Makron Books, 2000.

PORSCHE. **PORSCHE BRASIL**. Disponível em: <http://www.porsche.com.br/>. Acesso em 10 de set. 2007.

REESE, George. **JDBC e Java**: programação para banco de dados. 2.ed São Paulo: Berkeley, 2001.

RUMBAUGH, James; JACOBSON, Ivar. **UML:** guia do usuário. Rio de Janeiro: Ed. Campus, 2000.

SICA, Carlos. **PHP orientado a objetivos:** fale a linguagem da internet. Rio de Janeiro: Ciência Moderna, 2006.

THE PHP GROUP. **PHP:** hypertext preprocessor. Disponível em: <http://php.net>. Acesso em: 13 out. 2007.

XOOPS PROJECT. **Official XOOPS Website**. Disponível em: <http://www.xoops.org/>. Acesso em: 13 out. 2007.

W3C. **Web Content Accessibility Guidelines 1.0**. [S.l.: s.n.], 1999. Disponível em: <http://www.w3.org/TR/WAI-WEBCONTENT/>. Acesso em: 11 out. 2007.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos**. Rio de Janeiro: Elsevier, 2004.

WELLING, Luke, THOMSON, Laura. **PHP e MySQL**: desenvolvimento web. Rio de Janeiro: Campus, 2003.

ZANDSTRA, Matt. **Entendendo e Dominando o PHP:** Construa aplicações profissionais usando esta poderosa linguagem!. São Paulo: Apress, 2006.